Measuring Inheritance Coupling in Object-Oriented Systems

by

Mahmoud Omar Elish

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

December, 1999

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

MEASURING INHERITANCE COUPLING IN OBJECT-ORIENTED SYSTEMS

BY

MAHMOUD OMAR ELISH

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

DECEMBER 1999

UMI Number: 1398019

UMI®

UMI Microform1398019
Copyright 2000 by Bell & Howell Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by **MAHMOUD OMAR ELISH** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Thesis Committee

Dr. Jarallah S. AlGhamdi (Chairman)

Dr. Mostafa M. Aref (Member)

Dr. Moataz A. Ahmed (Member)

Dr. Jarallah S. AlGhamdi
Department Chairman

Dr. Abdullah M. Al-Shehri
Dean, College of Graduate Studies

18-12-99
Date


بسم الله الرحمن الرحيم

Dedicated To

My Parents and My Brothers

ACKNOWLEDGMENT

I would like to acknowledge King Fahd University of Petroleum and Minerals for all support extended during this research.

I would like to extend my appreciation to my thesis committee chairman, Dr. Jarallah AlGhamdi, for his continuous advice, guidance and cooperation. I feel grateful to my thesis committee members, Dr. Mostafa Aref and Dr. Moataz Ahmed, for their useful suggestions and cooperation.

Thanks to all colleagues and friends for their suggestions and encouragement during the preparation of this thesis

A very sincere appreciation to my father, my mother and my brothers for their prayers, encouragement and continuous support.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Name: Mahmoud Omar Elish

Thesis Title: Measuring Inheritance Coupling in Object-Oriented Systems

Major Field: Computer Science

Date of Degree: December 1999

The increasing importance being placed on software quality has led to a large number of new measures being proposed for quality design principles such as coupling. Given the importance of object-oriented development techniques, object-oriented metrics are needed to measure different aspects in object-oriented systems. Measuring class coupling is one way to evaluate the quality of such systems. In object-oriented design, three types of coupling may exist between classes: inheritance coupling, interaction coupling, and component coupling. Object-oriented metrics as other metrics need to be collected, stored, analyzed, and validated by suitable tools. However, there is lack of tools and thus developing tools to support these requirements is mandatory.

In this thesis, a tool for measuring inheritance coupling in object-oriented systems is developed. In addition, inheritance coupling is redefined in more appropriate way that takes into account indirect inheritance relationships. Some inheritance guidelines are also set to assist object-oriented designers in building good inheritance hierarchies. Moreover, a usability-based inheritance coupling metric is proposed. It considers the usability of inherited attributes and methods by inheriting classes in coupling calculation. Finally, two usability metrics are proposed: one measures the usage percentage of inherited elements from a class, and the other measures the usage percentage of inherited elements by inheriting class.

The developed tool is applied successfully against six case studies. Furthermore, it is compared and evaluated against three other existing tools. The comparison shows that the developed tool has some features that are not available in the other tools. It also indicates that the uniqueness of the developed tool relies on the framework it uses to calculate the coupling.

Keywords: Software Metrics, Coupling Metrics, Object-Oriented Metrics, Inheritance, Inheritance Coupling, Inheritance Guidelines, Interaction Coupling, Component Coupling, Usability Metrics, Measurement, Metrics Tools, Classes, Inheritance Hierarchies.

Master of Science Degree

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

December 1999

ARABIC ABSTRACT

خلاصة الرسالة

اســـم الـــطـــالـــب: محمود عمر عليش

عنـــوان الرســـالـــة: قياس الترابط التوارثي في أنظمة البرمجة بالذوات

الـــتـــخـــصـــص: علوم الحاسب الآلي

تاريـــخ الشـــهـــادة: ديسمبر ١٩٩٩م

أدى الاهتمام المتزايد بجودة برامج الحاسبات إلى ظهور عدد كبير من المعايير الجديدة لقياس جـــودة التصاميم ومنها الترابط بين أجزاء البرنامج. ونظراً لأهمية طرق البرمجة بالذوات فإن تطوير تقييـــم جودة هذه الأنظمة شيء ضروري. إن قياس ترابط فصائل هذه النظم هو أحد الوسائل لهذا التقييـــم، حيث يوجد ثلاثة أنواع من الترابط بين الفصائل: الترابط التوارثي و الترابط التفـــاعلي و الـــترابط بالمكونات. تحتاج طرق قياس البرامج إلى برامج مناسبة لتجميعها و تخزينها و تحليلـــها و تـــأكيد صحتها. وبما أن هناك نقص في هذه البرامج فإن تطوير برامج تدعم هذه المتطلبات أمر ضروري.

في هذه الرسالة تم تطوير برنامج لقياس الترابط التوارثي في أنظمـــة البرمجـــة بـــالذوات. بالإضافة إلى ذلك تم إعادة تعريف الترابط التوارثي بشكل أفضل متضمنا علاقات التـــوارث غـــير المباشرة. كما تم أيضاً وضع إرشادات لمساعدة مصممي أنظمة البرمجة بالذوات على بناء هيـــاكل توارثية على مستوى جيد. وقدمت الدراسة اقتراح وسيلة لقياس الترابط التوارثي عن طريق حســـاب استخدام الخصائص و الطرق المتوارثة في الفصائل الوارثة لها. كما تم أيضاً وضع طريقتين لقيـــاس الاستخدام، واحدة لقياس نسبة استخدام العناصر المتوارثة من الفصائل و الأخـــرى لقيـــاس نســـبة استخدام العناصر المتوارثة في الفصائل الوارثة لها.

وقد طبق البرنامج المطور بنجاح على عدة حالات دراسية، وتمت مقارنـــة و تقييـــم هـــذا البرنامج بثلاثة برامج أخرى، حيث أظهرت المقارنة أن البرنامج المطور له بعض المميزات التي لا تتوفر في البرامج الأخرى. كما أوضحت المقارنة أن البرنامج المطور ينفرد بالإطار الذي يســـتخدمه في قياس الترابط.

مفاتيح: طرق قياس البرامج، طرق قياس الترابط، طرق قياس أنظمة البرمجة بالذوات، التـــوارث، الترابط التوارثي، إرشادات التوارث، الترابط التفاعلي، الترابط بالمكونات، طرق قياس الاســـتخدام، مقاييس، أدوات القياس، الفصائل، الهياكل التوارثية.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول و المعادن
الظهران، المملكة العربية السعودية

ديسمبر ١٩٩٩م

CHAPTER 1

INTRODUCTION

Building quality systems is the primary goal of any software development organization. Most of software engineering efforts, within the last two decades, have been directed toward this goal [Eder94]. Software quality is a multidimensional concept that is mainly concerned with ensuring that software has low number of defects and that it reaches the required standards of maintainability, reliability, portability, and so on [Somm96]. The increasing importance being placed on software quality has led to a large number of new measures being proposed for quality design principles such as coupling [Chid94].

Object-oriented development techniques are popular concepts in today's software development. This popularity resulted from the promises of object-oriented paradigm about portability, reusability, maintainability, etc. [Bucc98]. Object-oriented development does not only require a different approach to design and implementation, but also requires a different approach to software

metrics [Rose98]. Given the importance of object-oriented development techniques, object-oriented metrics are needed to measure different aspects in object-oriented systems. Measuring class coupling is one way to evaluate the quality of object-oriented systems.

Following this increasing interest in object-oriented development is an increased interest in object-oriented metrics. Consequently, several object-oriented metrics have been proposed in the recent years. These metrics as other metrics need to be collected, stored, analyzed, and validated by suitable tools [Heri98]. However, there is lack of tools and thus developing tools to support these requirements is mandatory.

1.1 Concepts of Object-Oriented Systems

The first appearance of object-oriented programming was in late 1960s in SIMULA 67 [Sebe93]. The popularity of object-oriented paradigm has increased in the recent years due to its promises of portability, reusability, maintainability, etc. [Bucc98]. In the following subsections, the basic concepts of this paradigm are described.

1.1.1 Object and Object Class

An object is an encapsulation of state and behavior [Kors90]. In other words, it is an entity that consists of a set of instance variables (attributes) representing the internal state of the object and a set of operations (methods) representing the external behavior of the object [Eder94]. Each object represents an instance of some class. An object class is a template specifying state and behavior of a set of similar objects.

1.1.2 Inheritance

Inheritance is also known as "is-a" relationship. This concept is a unique contribution of the object-oriented paradigm [Kors90]. In object-oriented systems, classes are organized into a class hierarchy (inheritance tree) where subclasses inherit all the attributes and methods of their super-classes. Inheritance is transitive, i.e. a class can inherit features from super-classes many levels away. The subclasses can also have their own attributes and methods and they can override any of the inherited information. Multiple Inheritance is possible if a subclass has more than one super-class. Inheritance has two roles: an abstraction mechanism for classifying entities in system models, and a reuse mechanism for program code [Somm96].

Inheritance can affect the object-oriented design in both positive and negative ways. On one hand, inheritance does not only support reuse across system, but also facilitates extendibility within a given system [Kors90]. It also simplifies definition of classes similar to one(s) previously defined since the designers need to specify common attributes and methods once [Coad91]. Furthermore, the design with inheritance is represented in an abstract way that is easy to understand [Somm96].

On the other hand, inheritance spreads information around the design that may result in difficulty in understanding the design especially if the class hierarchy is deep [Somm96]. Moreover, misuse of inheritance may produce class hierarchies that are unstable and hard to maintain [Kuo94]. Therefore, inheritance should be properly applied to benefit from its advantages and avoid its disadvantages. Some inheritance guidelines are presented in section 3.3.3.

1.1.3 Message Passing, Polymorphism and Dynamic Binding

Objects communicate with each other by message passing. A message is a request to perform a certain operation (service) provided by the receiving objects. Polymorphism is another concept of object-oriented systems. It means that the same method may be invoked on objects of different classes [Sebe93]. Polymorphism is closely related to inheritance because the same operations

that apply to instances of a parent class also apply to instances of its subclasses [Kim89]. The binding between method invocation and code to be executed takes place during run-time and depends on the actual class of the object on which the method is invoked [Eder94]. This is known as dynamic binding.

1.2 Objectives of the Research

The main objectives of this research are the following:

(1) Developing a tool for measuring inheritance coupling in object-oriented systems to:

- Automatically gather, evaluate, and analyze inheritance coupling metrics.
- Simplify the inheritance coupling metrics data collection and ensure the accuracy and consistency of the collected data.
- Provide a platform for analyzing and comparing different inheritance coupling metrics.
- Provide a general system that is not limited to particular language and support for new object-oriented languages and metrics can be added.

(2) Setting some inheritance guidelines to assist object-oriented designers in building good inheritance hierarchies.

1.3 Motivation of the Research

This research was motivated by the following reasons:

- Metrics calculations can be complex and manual calculations can make metrics undesirable [Bris96].
- Presently, the commercial assessment tools are only capable of estimating a limited number of metrics [Bucc98].
- A tool for defining, showing, and validating object-oriented metrics is mandatory to manage this large number of metrics [Fior98].
- In order for metrics to become useful, they must be collected, stored, analyzed, and validated by suitable tool [Heri98].

1.4 Thesis Organization

The thesis is organized as follows. In chapter 2, software metrics are discussed. Chapter 3 covers coupling in object-oriented systems. The framework used in the developed tool is presented in chapter 4. Chapter 5 gives an overview of the developed tool as well as a comparison between it and three other existing tools. In chapter 6, case studies are presented. Finally, the conclusion and directions for future work come in chapter 7.

CHAPTER 2

SOFTWARE METRICS

A growing attention is given to the improvement of software development process by both designers and managers. Designers need to check the quality of their work, while managers need to estimate, monitor, and control the progress of the project [Mose97]. This has increased the demand for software metrics [Chid94]. Software metrics include wide range of activities such as cost estimation, quality control, performance evaluation, data collection, productivity measures, etc. [Fent93]. However in this thesis, they refer to any type of measurements which relate to a software system, process or related documentation [Somm96].

In this chapter, software metrics are briefly discussed. In section 2.1, the need for software metrics is explained. Desirable properties of software metrics are reviewed in section 2.2. In section 2.3, classifications of software metrics are presented. Object-oriented metrics are discussed in section 2.4 including their classification and existing object-oriented metrics suites.

2.1 The Need for Metrics

There is a clear need for software metrics in software development. According to DeMarco's principle [DeMa82]: "you cannot control what you cannot measure". Software metrics play an important role in evaluating and improving development processes and software products [Somm96]. In addition, they are useful in assessing the effort required in maintaining or re-engineering software systems [Snee95].

2.2 Desirable Properties

In order to be useful, software metrics should satisfy certain desirable properties. These include validity, reliability, and practical issues.

2.2.1 Validity

Metrics validation is the process of ensuring that the measure is a proper numerical characterization of the claimed attributes [Fent93]. It can be classified into two main categories: internal and external validation [Hend96]. Internal validity concerns with how well a measure captures real differences in the values of an attribute of the real world entities being measured. External validity, however, addresses the issue of generalizability, i.e. can the measure

be generalized beyond the sample entities measured and the environment in which the measurement took place?.

2.2.2 Reliability

Software metrics should be reliable [Hend96]. This can be achieved by satisfying two issues: stability and equivalence. A metric is stable if it produces the same results given the same entity in the same environment. The equivalence issue, on the other hand, addresses whether different samples affect the measurement results.

2.2.3 Practical Aspects

There are desirable properties of software metrics that are concerned with practical issues. For instance, metrics should be [Hend96]:

- Easy to apply and calculate;
- Automatible and provide timely feedback;
- Understandable and informative;
- Language independent; and
- Their data must be economical to collect.

2.3 Classifications of Software Metrics

Software metrics can be classified using different criteria. In [Somm96], they are classified into control metrics and predictor metrics. Control metrics are used to control the software process. Effort expended and elapsed time are two examples of control metrics. In contrast, predictor metrics are used to predict a product quality by measuring its attributes. For example, the complexity of software component can be predicted by measuring its cyclomatic complexity or line-of-code.

Fenton, in [Fent93], has classified software metrics into three classes based on entities whose attributes we may wish to measure. These classes are processes, products, and resources. Process metrics measures software related activities that normally have a time factor. Product metrics measures the output of software processes such as deliverables and documents. Resource metrics deal with input items to software processes. The attributes that are measured by these metrics are either internal or external. Internal attributes can be measured purely in term of their entities, while external attributes can only be measured with respect to how their entities are related to their environment. Table 2.1 gives examples of each metrics class.

Entities	Attribute	
	Internal	External
PRODUCTS		
Specifications	size, reuse, modularity, redundancy, functionality, syntactic correctness, etc.	comprehensibility, maintainability, etc.
Designs	size, reuse, modularity, coupling, cohesiveness, functionality, etc.	quality, complexity, maintainability, etc.
Code	size, reuse, modularity, functionality, algorithmic complexity, etc.	reliability, usability, maintainability, etc.
Test Data	size, coverage level, etc.	quality, etc.
...	...	...
PROCESSES		
Constructing Specification	time, effort, number of requirements changes, etc.	quality, cost, stability, etc.
Detailed Design	time, effort, number of specification faults found, etc.	cost-effectiveness, cost, etc.
Testing	time, effort, number of bugs found, etc.	cost-effectiveness, cost, stability, etc.
...	...	...
RESOURCES		
Personal	age, price, etc.	productivity, experience, etc.
Teams	size, communication level, etc.	productivity, quality, etc.
Software	price, size, etc.	usability, reliability, etc.
Hardware	price, speed, memory size, etc.	reliability, etc.
...	...	...

Table 2.1: Examples of process, product, and resource metrics.

2.4 Object-Oriented Metrics

The diffusion of object-oriented paradigm has led to the need for object-oriented metrics against which the object-oriented design can be evaluated. In object-oriented design, class is the basic entity of concern, not the procedure or statement. Hence, the metrics used to measure such design should be class-centric [Broo94].

2.4.1 A Classification of Object-Oriented Metrics

Object-oriented metrics can be classified into three levels of applicability [Bucc98, Fior98]: method-level metrics, class-level metrics, and system-level metrics. Method level metrics measure the complexity/size of methods. Traditional functional metrics can be used for this purpose such as line-of-code (LOC), Halstead measure, McCabe cyclomatic complexity, etc. [Fior98].

Class level metrics measure properties of the classes and their hierarchies in a design [Broo94]. These metrics are the most important object-oriented metrics because class is the basic entity of concern in object-oriented design [Broo94]. Examples of these metrics include number of methods per class (NOM), weight of methods per class (WOM), depth of inheritance tree (DIT), number of children (NOC), number of descendants (NOD), etc.

System level metrics measure the interactions of classes and objects on a system as whole [Broo94]. Mean method complexity, mean number of methods per class, maximum depth of inheritance, and number of trees are some examples of system level metrics.

2.4.2 Existing Object-Oriented Metrics Suites

Several object-oriented metrics have been developed in the recent years. In 1991, the object-oriented metrics suite by Chidamber and Kemerer [Chid91] was proposed. As shown in table 2.2, this suite contains six metrics focusing on size, coupling, and cohesion. A year after, three coupling complexity metrics and an interface metric, presented in table 2.3, were proposed by Henry and Li [Hend96]. In 1992, Tegarden and Sheetz suggested a large number of metrics listed in table 2.4 [Hend96]. These metrics covers variable level, method level, object level, and system level. Eleven design metrics, shown in table 2.5, were proposed by Lorenz in 1993 [Lore93]. In 1994, Lorenz and Kidd's metrics suite was developed [Lore94]. It contains over 30 different product/design metrics as presented in table 2.6.

Metric	Item Measured
Weight methods per class (WMC)	Size and complexity
Depth of inheritance tree (DIT)	Size
Number of children (NOC)	Size/coupling/cohesion
Coupling between objects (CBO)	Coupling
Response for class (RFC)	Communication and complexity
Lack of cohesion in methods (LCOM)	Internal cohesion

Table 2.2: Chidamber and Kemerer's metrics suite.

Coupling Complexity
Coupling through inheritance: DIT (depth of inheritance tree) + NOC (number of children)
Coupling through message passing: MPC = message-passing coupling
Coupling through data abstraction: DAC = data abstraction coupling
Interface Metric
NOM = number of methods per class

Table 2.3: Henry and Li's metrics suite.

Variable Level	**System Level**
Fan-In	Number of classes
Fan-Out	Number of abstract classes
Fan-Down	Number of concrete classes
Polymorphism	Max depth/breadth of inheritance hierarchy
Method Level	Max depth/breadth of uses hierarchy
Fan-In/fan-out/fan-down	Max depth/breadth of message graph
Method size (LOC)	Number of inheritance/uses links
Message path length	Number of unique messages sent between objects
Polymorphism	Number of root objects
Arguments passed to the methods	Average/max fan-in/fan-out
Arguments passed by the methods	Average/max fan-up/fan-down
Object Level	
Fan-In/fan-out	
Used-by/uses	
Arguments passed to object (methods)	
Arguments passed by object (methods)	
Fan-Up/fan-down	
Object-to-root depth	
Object-to-leaf depth	
Number of local variables/methods	
Number of inherited variables/methods	
Number of subclass variables/methods	
Average/max path length	

Table 2.4: Tegarden and Sheetz's metrics suite.

Metric	Item Measured
Average method size	Size
Average number of methods per class	Size
Average number of instance variables per class	Coupling
Class hierarchy nesting level (or DIT)	Size
Number of subsystem/subsystem relationship	Coupling
Number of class/class relationships within each subsystem	Cohesion and coupling
Instance variable usage	Semantic complexity
Average number of comment lines	Cognitive complexity
Number of problems reports per class	Process
Number of times class id reused	Process
Number of classes and methods thrown away	Process

Table 2.5: Lorenz's metrics suite.

Method Size	**Method Inheritance**
Number of message sends	Number of methods overridden by a subclass
Number of statements	Number of methods inherited by a subclass
Lines of code	Number of methods added in a subclass
Average method size	Specialization index
Method Internals	**Class Internals**
Method complexity	Class cohesion
Strings of message sends	Global usage
Class Size	Average number of parameters per method
Number of public instance methods per class	Use of friend functions
Number of instance methods per class	Percentage of function-oriented code
Average number of instance methods per class	Average number of comment lines per method
Number of instance variables per class	Average number of commented methods
Average number of instance variables per class	Number of problem reports per class or contract
Number of class methods per class	**Class Externals**
Number of class variables per class	Class coupling
Class Inheritance	Number of times a class is reused
Class hierarchy nesting level	Number of classes/methods thrown away
Number of abstract classes	
Use of multiple inheritance	

Table 2.6: Lorenz and Kidd's metrics suite.

CHAPTER 3

COUPLING IN OBJECT-ORIENTED SYSTEMS

The concept of coupling was introduced by Stevens et al. [Stev74] in 1974. It is defined as a measure of the strength of component interconnections [Somm96], i.e., the degree of interdependence between components [Fent93]. In object-oriented design (OOD), coupling is the interconnectedness between pieces (classes) of an OOD [Coad91]. Coupling is one of the design quality attributes as far as complexity and maintainability of a design are concerned. Loose coupling is desirable in order to:

1. Localize the effect of change where a change in a component affect another component as little as possible;
2. Reduce the chance that a fault in one component of the system will cause a failure in the other components; and
3. Increase the understandability of the software system.

This chapter discusses the three types of coupling that may exist between classes in object-oriented systems. In section 3.1, interaction coupling is

explained. Component coupling is presented in section 3.2. Section 3.3 discusses inheritance coupling. Finally, object-oriented coupling measurements are reviewed in section 3.4.

3.1 Interaction Coupling

In object-oriented systems, objects communicate with each other via message passing. This type of communication introduces the interaction coupling which is a coupling between two objects expressed by message connection [Coad91]. Low interaction coupling is desirable since it promotes encapsulation, simplifies testing, and enforces reusability of classes [Chid94, Coad91].

Interaction coupling is similar to the original definition of coupling [Eder94]. Accordingly, some degrees of classical coupling, proposed by Myers [Myer78], can be used to describe interaction coupling as follows (from worst to best) [Budd91]:

- *Internal Data Coupling.* It occurs if one class is allowed to modify the local data values (instance variables) of another class.
- *Control (or Sequence) Coupling.* It occurs if one class must perform operations in a certain fixed order, but the order is controlled by another class.

- *Parameter Coupling.* It occurs if one class must invoke services (methods) of another class, and the only relationships are the number and type of parameters supplied and the type of value returned.
- *Nil.* No interaction coupling between two classes.

Definition 3.1: ***Interaction Coupling***

Given an object-oriented system with n classes $C_1, C_2, \ldots, C_n$, two classes C_i with a set of methods M_i and C_j with a set of methods M_j are interaction coupled if either:

1. C_j invokes a method m where $m \in M_i$; or
2. C_i invokes a method m where $m \in M_j$.

where $i \neq j$ and $1 \leq i, j \leq n$.

3.2 Component Coupling

A class can be used as domain of some instance variables of another class. This introduces the concept of component coupling between classes [Eder94]. Component coupling is defined in definition 3.2. There are four degrees of component coupling between classes listed below from strongest to weakest [Eder94]:

- *Hidden.* Component coupling does not manifest itself in code.
- *Scattered.* Component coupling manifests itself in the body of the class. This can occur if a class is the type of either a local variable of a method, or an input or output parameter of a method invoked within a method of another class.
- *Specified.* Component coupling manifests itself in the interface. This can happen if a class is the type of either an attribute, or an input or output parameter of a method of another class.
- *Nil.* No component coupling.

Definition 3.2: ***Component Coupling***

Given an object-oriented system with *n* classes C_1, C_2, ... , C_n, two classes C_i and C_j are component coupled if C_i is the type of either:

1. an attribute of C_j; or
2. an input or output parameter of a method of C_j; or
3. a local variable of a method of C_j; or
4. an input or output parameter of a method invoked within a method of C_j.

where $i \neq j$ and $1 \leq i, j \leq n$.

3.3 Inheritance Coupling

Inheritance coupling is a coupling between generalization classes (super-classes) and their specialization classes (subclasses) [Coad91]. Two classes are inheritance coupled if one is a direct or indirect subclass of the other [Eder94]. High inheritance coupling is desirable [Coad91]. To achieve this, each class should indeed be a specialization of its super-class. In other words, each class should maximally utilize the inherited attributes and methods by using them and minimizing the overriding. There are four degrees of inheritance coupling described below from highest to lowest [Eder94]:

- *Modification.* Inheriting class changes inherited method(s) in a manner that violates some predefined "good practice" rules. These rules state that "an inherited method must not be deleted from the class interface," and "if a method is overridden, the overriding method must keep the same semantics as the overridden method".
- *Refinement.* Inheriting class changes inherited method(s) according to the predefined "good practice" rules.
- *Extension.* Inheriting class only adds new attributes and methods, but neither modifies nor refines any of the inherited ones.
- *Nil.* No inheritance relationship between two classes.

It worth noting that there is no contradiction between the use of inheritance for gaining better reusability and the goal of low coupling [Eder94]. This can be further explained as follows [Eder94]:

1. Inheritance may be used to lower non-inheritance coupling in an object-oriented system through generalization. For instance, consider a class *B* which invokes the same method *m* on objects of classes *A1* and *A2*. *B* is interaction coupled to *A1* and *A2*. If the method *m* is factored out into a common super-class *A* of *A1* and *A2* and not overridden in *A1* and *A2*, *B* is interaction coupled with *A* only.
2. There are different degrees of inheritance coupling as described above. The modification and refinement inheritance coupling should be minimized, and the extension inheritance coupling should be maximized to improve reusability.

3.3.1 Redefining Inheritance Coupling

In the literature, inheritance coupling is defined as a coupling between generalization classes and their specialization classes [Coad91]. Two classes are inheritance coupled if one is a direct or indirect subclass of the other [Eder94]. According to this definition, there is no inheritance coupling between classes *B* and *C* shown in figure 3.1. However, there is indirect inheritance

relationship between classes *B* and *C* that cannot be ignored since they have the same super-class from which they inherit the same information. To illustrate, assume that class *C* is changed by adding the method *m* of class *B* to it. In this case, the method *m* needs to be removed from classes *B* and *C* and generalized into the common super-class *A*. This is just one example from many other possible situations. Therefore, it is more appropriate to extend the definition of inheritance coupling to include indirect inheritance relationships. Accordingly, inheritance coupling should be defined as coupling resulted from inheritance between any two classes in the same inheritance tree.

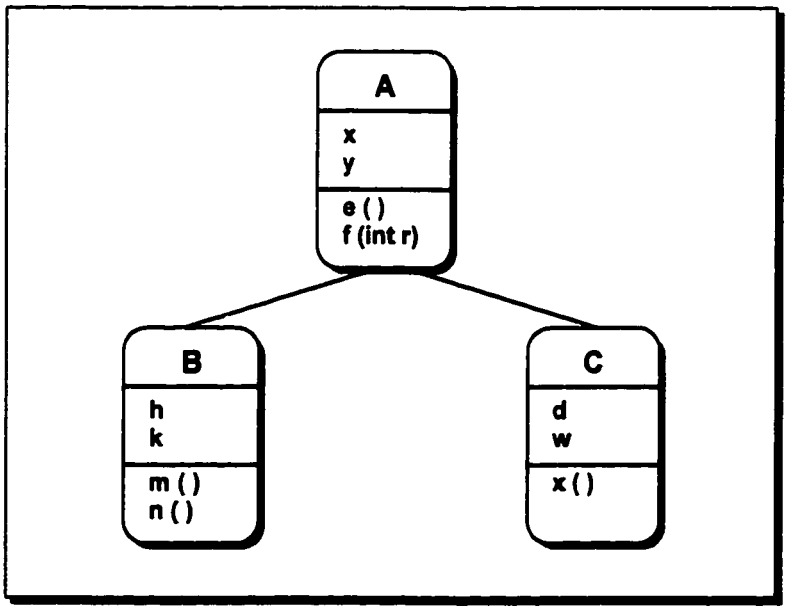


Figure 3.1: Inheritance hierarchy example.

Definition 3.3: ***Shared Attribute between Two Classes***

Given an object-oriented system with n classes $C_1, C_2, \ldots, C_n$, an attribute a is a shared attribute between two classes C_i and C_j if:

1. $\exists C_k$ with a set of attributes A_k where $a \in A_k$ and C_k is a direct or indirect super-class of both C_i and C_j; and
2. a is inherited and used by both C_i and C_j.

where $i \neq j \neq k$ and $1 \leq i, j, k \leq n$.

Definition 3.4: ***Shared Method between Two Classes***

Given an object-oriented system with n classes $C_1, C_2, \ldots, C_n$, a method m is a shared method between two classes C_i and C_j if:

1. $\exists C_k$ with a set of methods M_k where $m \in M_k$ and C_k is a direct or indirect super-class of both C_i and C_j; and
2. m is inherited and used by both C_i and C_j.

where $i \neq j \neq k$ and $1 \leq i, j, k \leq n$.

Definition 3.5: ***Inheritance Coupling***

Given an object-oriented system with n classes $C_1, C_2, \ldots, C_n$, two classes C_i and C_j are inheritance coupled if either:

1. C_i is a direct or indirect subclass of C_j; or
2. C_j is a direct or indirect subclass of C_i; or
3. $\exists a$ such that a is a shared attribute between C_i and C_j; or
4. $\exists m$ such that m is a shared method between C_i and C_j.

where $i \neq j$ and $1 \leq i, j \leq n$.

3.3.2 Contributing Factors

There are different factors contributing to the value of inheritance coupling. Seven major factors are discussed in this section.

1. ***Number of methods per class (NOM)***

   The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class [Chid94].

2. ***Number of attributes per class (NOA)***

   Similarly, as in NOM, the larger the number of attributes in a class the greater the potential impact on children, since children will inherit all the attributes defined in the class [Almu98].

3. ***Weight of methods per class (WOM)***

The design complexity metrics, such as LOC, can be used to measure the weight of the methods in a class. The complexity of a class increases when its methods gain more weight [Chid94], and this in turn impacts all descendant subclasses and affects the degree of inheritance coupling accordingly.

4. ***Weight of attributes per class (WOA)***

Similarly, as in WOM, the complexity of a class increases when its attributes gain more weight [Almu98]. Therefore the degree of inheritance coupling will be affected.

5. ***Depth of inheritance tree (DIT)***

This is defined to be the length of the longest path from the class to the root class in the inheritance tree. The impact of DIT on inheritance coupling is explained in [Chid94] as follows:

1) The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior.
2) Deeper trees constitute greater design complexity, since more methods and classes are involved.

3) The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

6. *Number of children (NOC)*

This is defined to be the number of immediate subclasses. The effect of NOC is explained in [Chid94] as follows:

1) Greater the number of children, greater the reuse, since inheritance is a form of reuse.
2) Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of subclassing.
3) The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

7. *Number of descendants (NOD)*

This is defined to be the number of all descendant children, immediate and non-immediate. It is important since the number of immediate children might be so small compared to the number of non-immediate children [Almu98].

3.3.3 Inheritance Guidelines

It is well known that loose coupling is desirable. However, inheritance coupling is a special case in the sense that neither high nor low inheritance coupling is desirable because inheritance can affect the object-oriented design in both positive and negative ways as discussed in section 1.1.2. Instead, inheritance should be properly applied. The following inheritance guidelines aim to assist object-oriented designers in building good inheritance hierarchies.

- The impact of depth of inheritance on maintainability was studied through experiment in [Daly96]. The experimental results showed that maintaining object-oriented software with three levels of inheritance is quicker than maintaining equivalent object-based software without inheritance. In contrast, maintaining object-oriented software with five levels of inheritance takes longer time than maintaining equivalent object-based software without inheritance.

- If a super-class has only one subclass, then the designer should closely look at the relationship between them, and whether the inheritance relationship is required or not [Almu98].

- The modification and refinement inheritance coupling should be minimized, and the extension inheritance coupling should be maximized [Eder94].

- Each class should maximally utilize the inherited attributes and methods by using them and minimizing the overriding [Coad91].

3.4 Object-Oriented Coupling Measurements

In the recent years, several object-oriented coupling metrics have been proposed. They measure the three types of coupling in object-oriented systems that are discussed in the previous sections. This section aims to describe briefly the object-oriented coupling metrics presented in the literature.

- ***Coupling between objects (CBO)***

  CBO for a class is defined in [Chid91] as a count of the number of noninheritance related couples with other classes. In [Chid94], the definition is revised to a count of the number of other classes to which a class is coupled. This includes coupling due to inheritance.

- ***Coupling through inheritance (IC)***

  In 1993, Li and Henry proposed a metric that measures the coupling through inheritance [Li93]. The proposed metric is simply equal to the

depth of inheritance tree *(DIT)* plus the number of children *(NOC)*.

$$Coupling\ through\ inheritance = DIT + NOC \qquad (3.1)$$

- ***Message passing coupling (MPC)***

  This metric was suggested by Li and Henry [Li93]. It is equal to the number of send statements defined in a class.

- ***Data abstraction coupling (DAC)***

  This metric is proposed in [Li93] as the number of abstract data types (ADTs) defined in a class.

- ***Class coupling metric (CC)***

  This metric was proposed by Chen and La [Chen93]. It measures the coupling between a class and other classes by summing the following:

  1. The number of access to other classes;
  2. The number of access by other classes; and
  3. The number of co-operated classes. A co-operated class is a class in which a method requests some services from another class and vice versa.

- ***AlMulla***

  In 1998, AlMulla proposed an inheritance coupling metric [Almu98]. The

seven factors considered by this metric are described in section 3.3.2. This metric provides three different coupling measurements: individual class coupling, class-to-class coupling, and overall system coupling. The calculation process of this metric is based on the framework presented in the next chapter.

The following table illustrates the type(s) of coupling measured by each of the above metric.

Metric	Coupling Measured		
	Component	Interaction	Inheritance
CBO		✓	✓
IC			✓
MPC		✓	
DAC	✓		
CC		✓	
AlMulla			✓

Table 3.1: Coupling measured by presented metrics.

CHAPTER 4

FRAMEWORK

The framework used in the developed tool is based on the concept of cover-coefficient-based clustering methodology for text databases introduced by Can and Ozkarahan in [Can90]. This concept was enhanced by AlGhamdi in [Algh94] by introducing it to software metrics, and therefore make it possible to generate a coupling matrix that represents coupling degrees between the components of software systems. The framework calculates the coupling in two steps as illustrated in figure 4.1. The first step aims to generate a description of the system that captures all characteristics affecting the coupling of software system. The representation created is called definition matrix. In the second step, coupling matrix is automatically generated from the definition matrix using the coupling formula presented by Can and Ozkarahan in [Can90].

In this chapter, the framework used in the developed tool is described in details from the inheritance coupling point of view as shown in figure 4.2. The

calculation of both definition matrix and coupling matrix is discussed in sections 4.1 and 4.2 respectively. In section 4.3, an example that illustrates the applicability of the framework is given. Section 4.4 covers usability-based inheritance coupling calculation. Finally, the advantages of the framework are explained in section 4.5.

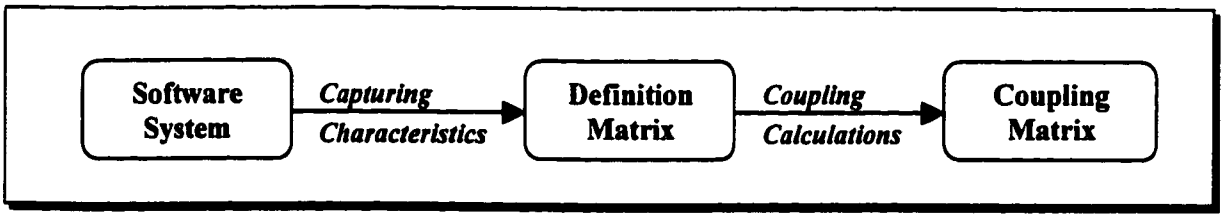


Figure 4.1: Framework steps.

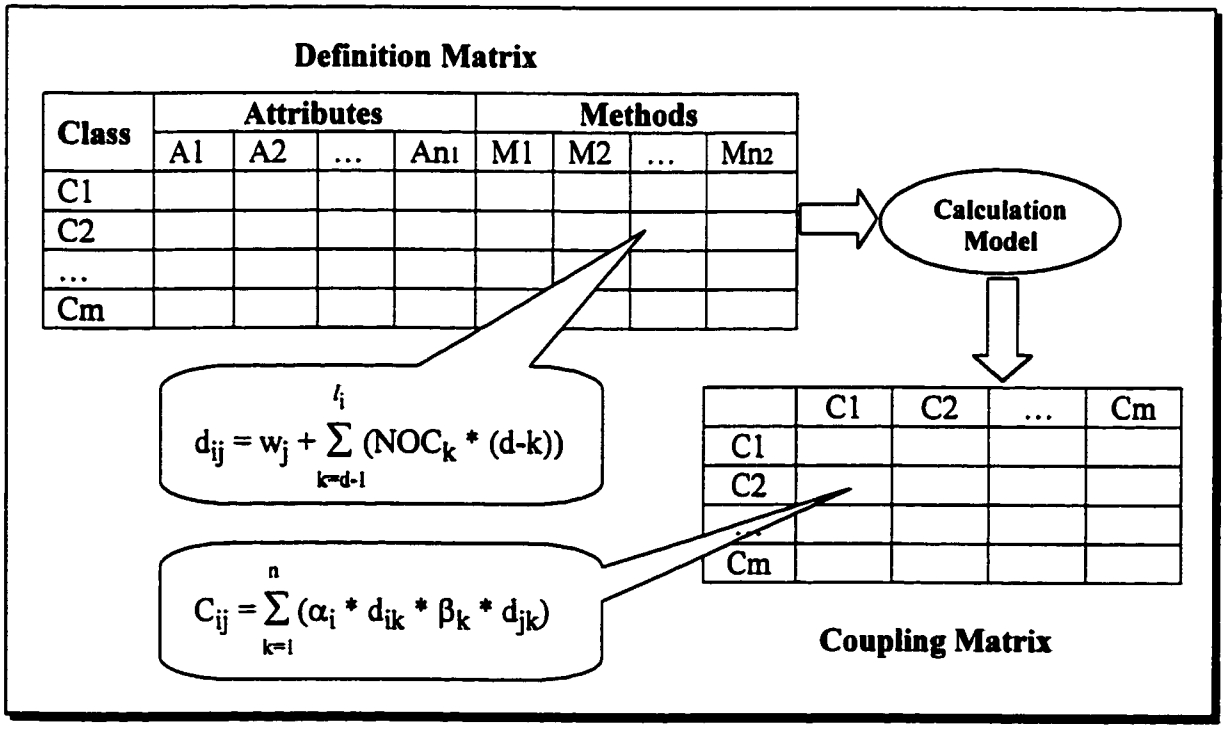


Figure 4.2: The framework from the inheritance coupling point of view.

4.1 Definition Matrix

System components (i.e. classes, attributes, and methods) are represented on a data structure called the definition matrix. Each row in the definition matrix represents a class of the system, while each column in the definition matrix represents an element (attribute or method) of the system. Table 4.1 illustrates the definition matrix.

Class	Attributes				Methods			
	A1	A2	...	An1	M1	M2	...	Mn2
C1								
C2								
...								
Cm								

Table 4.1: Definition matrix.

Each entry d_{ij} in the definition matrix shows the weight of element j corresponding to class i. A zero value in d_{ij} entry in the definition matrix indicates the absence and the inaccessibility of element j from class i. An entry in the definition matrix can be calculated using the following formula that was proposed by AlMulla in [Almu98]:

$$d_{ij} = w_j + \sum_{k=d-1}^{l_i} (NOC_k * (d-k)) \qquad (4.1)$$

Where,

w_j represents the j^{th} attribute size or method complexity;

l_i represents the level of class *i* in the inheritance subtree rooted by class *i*;

d represents the depth of inheritance (DIT) for the subtree rooted by class *i*, where the DIT is defined as the longest path from the root, i.e. class *i*, to any of the leaves in the subtree; and

NOC_k represents the number of children that are descendant from level *k* and inherit the j^{th} attribute or method through class *i*.

The definition matrix represents the basis from which the coupling values can be automatically generated as described in section 4.2. The accuracy of the definition matrix depends on the weighing scheme used to assign weights to the factors that influence coupling.

4.1.1 Attributes Weighing Scheme

In order to assign weights to the attributes, Chen and La's attribute complexity metric is adopted [Chen93]. This metric, as shown in table 4.2, is based on attribute size, i.e. the smaller the data type the smaller its complexity value and vice versa. Since the weight must be greater than zero, boolean and integer attributes will be assigned a complexity value of one. Data types that are not listed in table 4.2 can be treated similarly. It should be clear that these

complexity values given in table 4.2 are the default values which are subject to tuning as future work.

Type	Complexity Value
Boolean or integer	0
Char	1
Real	2
Array	3-4
Pointer	5
Record, Struct or Object	6-9
File	10

Table 4.2: Attribute complexity metric.

4.1.2 Methods Weighing Scheme

The weighing scheme used to calculate the methods complexity was proposed by AlMulla in [Almu98] as follows:

$$\text{method complexity} = \lceil LOC_{method} / LOC_{avg} \rceil \qquad (4.2)$$

Where,

LOC_{method} represents the method Line-Of-Code; and

LOC_{avg} represents the average Line-Of-Code of all methods in the system.

4.2 Coupling Matrix

The coupling matrix, as shown in table 4.3, has equal number of rows and columns, where each class is represented by one row and one column. It can be calculated easily using the definition matrix according to the following formula presented in [Can90]:

$$C_{ij} = \sum_{k=1}^{n} (\alpha_i * d_{ik} * \beta_k * d_{jk}) \qquad \text{for i and j} = 1..m \qquad (4.3)$$

Where,

m the number of classes in the system;

n the number of elements in the definition matrix ($n = n1 + n2$);

d_{ik}, d_{jk} elements of the definition matrix;

α_i the inverse of the sum of the i^{th} row of the definition matrix; and

$$\alpha_i = \left(\sum_{j=1}^{n} d_{ij} \right)^{-1}$$

β_k the inverse of the sum of the k^{th} column of the definition matrix.

$$\beta_k = \left(\sum_{i=1}^{m} d_{ik} \right)^{-1}$$

	C1	C2	...	Cm
C1				
C2				
...				
Cm				

Table 4.3: Coupling matrix.

4.2.1 Properties

The following properties hold for the coupling matrix [Can90]:

1. $0 \leq C_{ij} \leq 1$
2. The sum of each row equals 1
3. If $C_{ij} = 0$ then $C_{ji} = 0$, and similarly if $C_{ij} > 0$ then $C_{ji} > 0$; but in general,

   $C_{ij} \neq C_{ji}$
4. $C_{ii} = C_{jj} = C_{ij} = C_{ji}$ iff d_i and d_j are identical.

The above properties can be further explained as follows [Algh94]:

Property 1:

a. $C_{ij} = 0$, if: $\forall$ k ($d_{jk} > 0 \Rightarrow d_{ik} = 0$);

b. $C_{ii} = 1$, if: ($\forall$ k ($d_{jk} > 0 \Rightarrow \exists$ not ($d_{jk} > 0$ where $i \neq j$))); and

c. $0 < C_{ij} < 1$, $\forall$ k ($\exists$ ($d_{ik} > 0$) and $\exists$ ($d_{jk} > 0$) where $i \neq j$).

Property 2:

for all classes, the row sum of the coupling matrix equals one.

Property 3:

a. $(C_{ij} = 0) \Rightarrow (C_{ji} = 0)$;

b. $(C_{ij} > 0) \Rightarrow (C_{ji} > 0)$;

c. there may exist $C_{ij} = C_{ji}$; and

d. there may exist $C_{ij} \neq C_{ji}$.

Property 4:

$\forall k\ (d_{ik} = d_{jk}) \Rightarrow C_{ii} = C_{jj} = C_{ij} = C_{ji}$.

4.2.2 Coupling Measurements

Once the coupling matrix is constructed, three coupling measurements can be obtained: individual class coupling, class-to-class coupling, and overall system coupling.

- **Individual Class Coupling (ICC)**

  The coupling of class i, which shows the total connections that class i has with other classes of the system, can be calculated as:

$$C_i = 1 - C_{ii} \qquad (4.4)$$

- **Class-To-Class Coupling (CTC)**

  Each entry C_{ij} of the coupling matrix represents the extent to which class i is coupled to class j.

- **Overall System Coupling (OSC)**

  The coupling of the whole object-oriented system is the average of the coupling of all classes of the system, which shows the extent to which the classes of the system are connected to each other.

$$OSC = \left(\sum_{i=1}^{m} C_i \right) / m \qquad (4.5)$$

4.3 Example

Consider a hypothetical system with seven classes named C1, C2, C3, C4, C5, C6, and C7. Figure 4.3 shows the inheritance hierarchy diagram for this system. The attributes and methods details for each class are given in table 4.4. Attributes size in table 4.4 are determined from table 4.2, whereas methods complexity are calculated using formula 4.2. For example,

The complexity of method M21(a) = $\lceil 10/8.1 \rceil = 2$

where, $LOC_{avg} = 81/10 = 8.1$

There are 7 classes with a total of 21 elements (11 attributes and 10 methods) in this system. Therefore, 7 X 21 definition matrix will be constructed as shown in table 4.5. Each entry in the definition matrix is calculated using formula 4.1. For example,

$$d_{12} = 5 + (3 * (2-1) + 3 * (2-0)) = 14$$

The coupling matrix of the hypothetical system is shown in table 4.6. Each entry in the coupling matrix is calculated using formula 4.3. Class-to-class coupling (CTC), individual class coupling (ICC), and overall system coupling (OSC) are calculated as described in section 4.2.2, and the results are shown in tables 4.6 and 4.7.

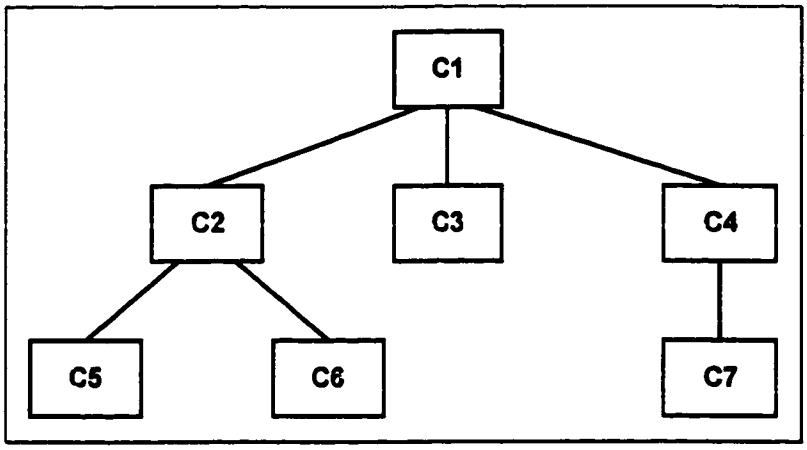


Figure 4.3: Inheritance hierarchy diagram for the hypothetical system.

Classes	Attributes			Methods		
	Name	Type	Size	Signature	LOC	Complexity
C1	a b	integer pointer	1 5	M11(a)	8	1
C2	c	file	10	M21(a,c) M22(c)	10 3	2 1
C3	d e	char boolean	1 1	M31(d,e)	5	1
C4	f	array	4	M41(f,b)	12	2
C5	g h	integer char	1 1	M51(a,c,g) M52(h)	7 20	1 3
C6	i j	struct pointer	8 5	M61(i) M62(c,j)	4 6	1 1
C7	k	real	2	M71(k)	6	1

Table 4.4: Classes attributes and methods of the hypothetical system.

	Attributes											Methods										
	a	b	c	d	e	f	g	h	i	j	k	M11	M21	M22	M31	M41	M51	M52	M61	M62	M71	α
C1	10	14	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0.03
C2	3	7	12	0	0	0	0	0	0	0	0	3	4	3	0	0	0	0	0	0	0	0.03
C3	1	5	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0.10
C4	2	6	0	0	0	5	0	0	0	0	0	2	0	0	0	3	0	0	0	0	0	0.06
C5	1	5	10	0	0	0	1	1	0	0	0	1	2	1	0	0	1	3	0	0	0	0.04
C6	1	5	10	0	0	0	0	0	8	5	0	1	2	1	0	0	0	0	1	1	0	0.03
C7	1	5	0	0	0	4	0	0	0	0	2	1	0	0	0	2	0	0	0	0	1	0.06
β	0.05	0.02	0.03	1.00	1.00	0.11	1.00	1.00	0.13	0.20	0.50	0.05	0.13	0.20	1.00	0.20	1.00	0.33	1.00	1.00	1.00	

Table 4.5: Definition matrix of the hypothetical system.

	C1	C2	C3	C4	C5	C6	C7
C1	0.4323	0.1542	0.0748	0.1145	0.0748	0.0748	0.0748
C2	0.1638	0.3216	0.0331	0.0477	0.2003	0.2003	0.0331
C3	0.2542	0.1060	0.3637	0.0849	0.0637	0.0637	0.0637
C4	0.2162	0.0847	0.0472	0.3203	0.0472	0.0472	0.2373
C5	0.0978	0.2466	0.0245	0.0326	0.4024	0.1716	0.0245
C6	0.0726	0.1832	0.0182	0.0243	0.1275	0.5561	0.0182
C7	0.1589	0.0663	0.0398	0.2669	0.0398	0.0398	0.3884

Table 4.6: Coupling matrix of the hypothetical system.

Class	Inheritance Coupling
C1	0.5677
C2	0.6784
C3	0.6363
C4	0.6797
C5	0.5976
C6	0.4439
C7	0.6116
Overall System Coupling (OSC)	*0.6002*

Table 4.7: Individual classes coupling (ICC) of the hypothetical system.

4.4 Usability-Based Inheritance Coupling Calculation

The inheritance coupling metric proposed by AlMulla does not take into account whether inherited attributes and methods are used by the inheriting classes or not [Almu98]. However, considering the usability of inherited elements in inheritance coupling calculation is essential since this what actually makes the coupling. Accordingly, to calculate the usability-based inheritance coupling, the definition matrix of AlMulla metric described in section 4.1 needs to be replaced by another one called the usability-based definition matrix. This new matrix has same size as the corresponding definition matrix of AlMulla metric. The only difference is in the formula used to calculate each entry in the matrix. An entry in the usability-based definition matrix can be calculated using the following formula:

$$d_{ij} = \left(w_j + \sum_{k=d-1}^{l_i} (NOC_k * (d-k)) \right) * U \qquad (4.6)$$

Where,

w_j represents the j^{th} attribute size or method complexity;

l_i represents the level of class *i* in the inheritance subtree rooted by class *i*;

d represents the depth of inheritance (DIT) for the subtree rooted by class *i*, where the DIT is defined as the longest path from the root, i.e. class *i*, to any of the leaves in the subtree;

NOC_k represents the number of children that are descendant from level *k* and inherit the j^{th} attribute or method through class *i*; and

$$U = \begin{cases} 1 & \text{If element } j \text{ is inherited from class } i \text{ and used in any of its descendant classes, or if element } j \text{ is inherited by class } i \text{ and used in it or in any of its descendant classes.} \\ 0 & \text{Otherwise.} \end{cases}$$

Multiplication by U is the only difference between formula 4.1 and formula 4.6. This means that each entry in the usability-based definition matrix is equal to either the corresponding entry in the definition matrix of AlMulla metric or zero. In object-oriented systems, these two matrices are the same if all inherited attributes and methods are used by the inheriting classes. Once the usability-based definition matrix is created, the usability-based coupling matrix can be easily generated as described in section 4.2. Three different coupling measurements can be obtained from this coupling matrix: usability-based individual class coupling (UICC), usability-based class-to-class coupling (UCTC), and usability-based overall system coupling (UOSC).

4.5 Advantages of the Framework

The advantages of the presented framework can be summarized as follows:

- The framework is not limited to object-oriented paradigm, and it can be applied easily to other paradigms by adjusting formula 4.1.
- The framework breaks the measurements of coupling into two steps. This does not only simplify coupling measurement, but also enables the researchers to concentrate in each step separately and try to improve it.
- The framework, as discussed in section 4.2.2, provides three different coupling measurements: individual class coupling, class-to-class coupling, and overall system coupling.

CHAPTER 5

DEVELOPED TOOL: AN OVERVIEW & EVALUATION

Several object-oriented coupling metrics have been proposed in the recent years. These metrics need to be supported by suitable tools for automatically collecting, storing, analyzing, and validating them [Heri98]. However, there is lack of tools, and thus an automated tool is mandatory to support these requirements [Fior98].

A tool for measuring inheritance coupling in object-oriented systems was developed. It works as follows. First, it parses object-oriented systems to collect inheritance coupling metrics data. The collected data are then abstracted into a language independent format according to a conceptual model, and stored in the system's database. Finally, query engines can be applied on the database to calculate the coupling metrics. Although the tool is currently supporting Java software, it is not limited to particular language and support for new object-oriented languages and metrics can be added easily.

This chapter gives an overview and evaluation of the developed tool.

Tool structure is described in section 5.1. In section 5.2, tool implementation is discussed. Section 5.3 describes the metrics supported by the developed tool. Features of the developed tool are listed in section 5.4. Section 5.5 presents three other existing tools. Finally, a comparison between the developed tool and existing tools is given in section 5.6.

5.1 Tool Structure

The developed tool consists of three main components: parsing engines, central metrics repository, and query engines. Each one of them is briefly described in the following subsections. Figure 5.1 illustrates the structure of the tool.

5.1.1 Parsing Engines

Data collection is the first step in calculating any software metrics. Parsing engines aim to extract the information needed to compute inheritance coupling metrics from either object-oriented designs or software. These parsers are language dependent, i.e. a separate parser need to be implemented for each object-oriented language like Java, C++, Smalltalk, etc. and each object-oriented design tool such as UML (Unified Modeling Language). The data collected by these parsers are abstracted into a language independent format according to a conceptual model formed with ER model shown in figure 5.2.

Therefore, the output of the parsing engines must have the same format and structure. In this work, a Java parser was developed. It can parse any Java source code to collect inheritance coupling metrics data into a text file. The tool used in implementing this parser is described in section 5.2.

5.1.2 Central Metrics Repository

The data collected by the parsing engines are loaded into the central metrics repository of the tool in which they are stored for later use. This repository was designed based on the ER model presented in figure 5.2. This ER diagram models the object-oriented system from inheritance point of view. In addition, this model is language independent, i.e. it represents an abstract format of object-oriented system. The central metrics repository provides the base for all metrics calculation since all metrics will retrieve their input parameters from it.

5.1.3 Query Engines

Query engines are functions or SQL that can be applied on the central metrics repository to calculate the coupling metrics. General purpose functions were developed to simplify and speed up the creation of new query engine. These general purpose functions are common functions used by different calculation procedures.

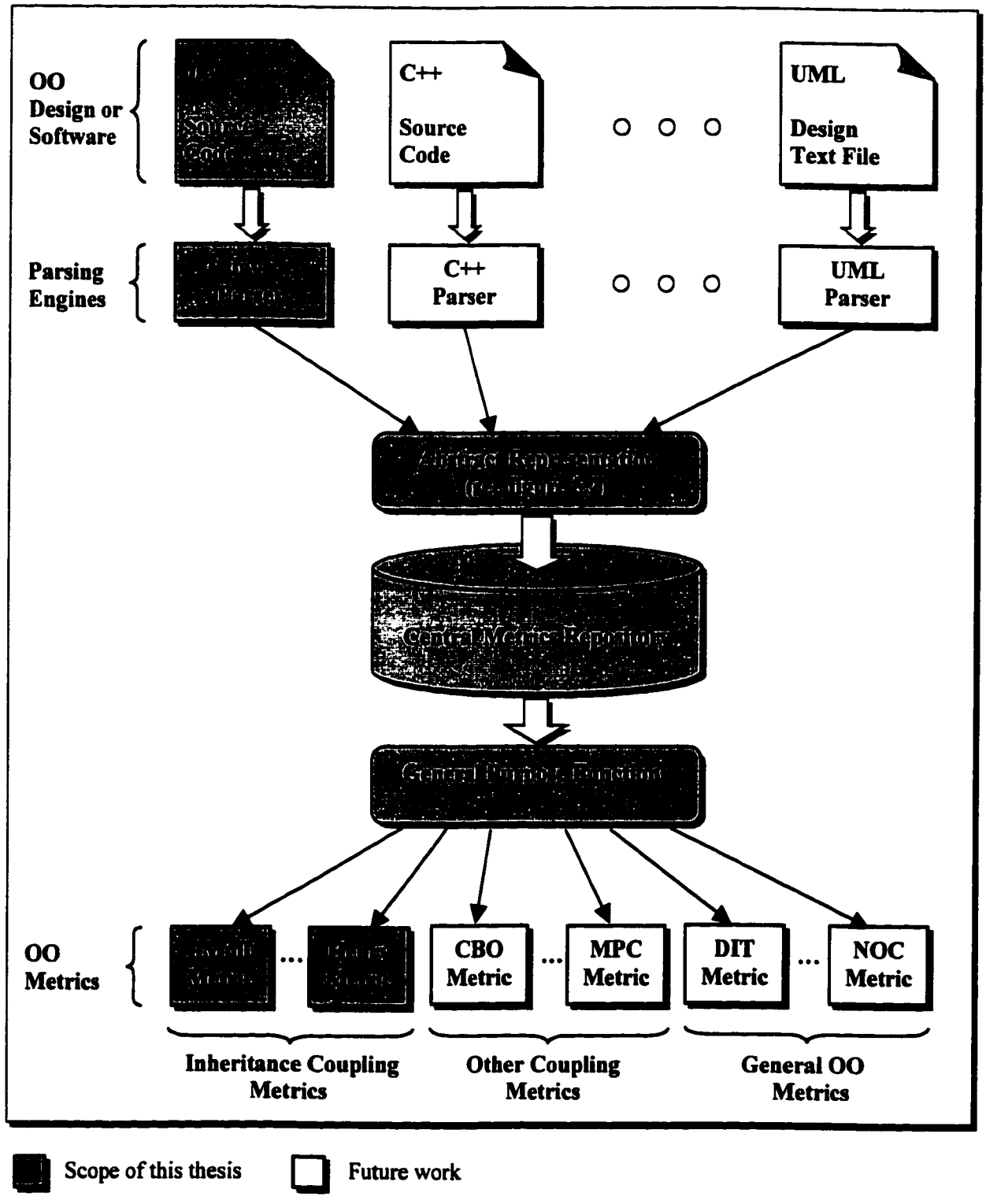


Figure 5.1: Tool structure.

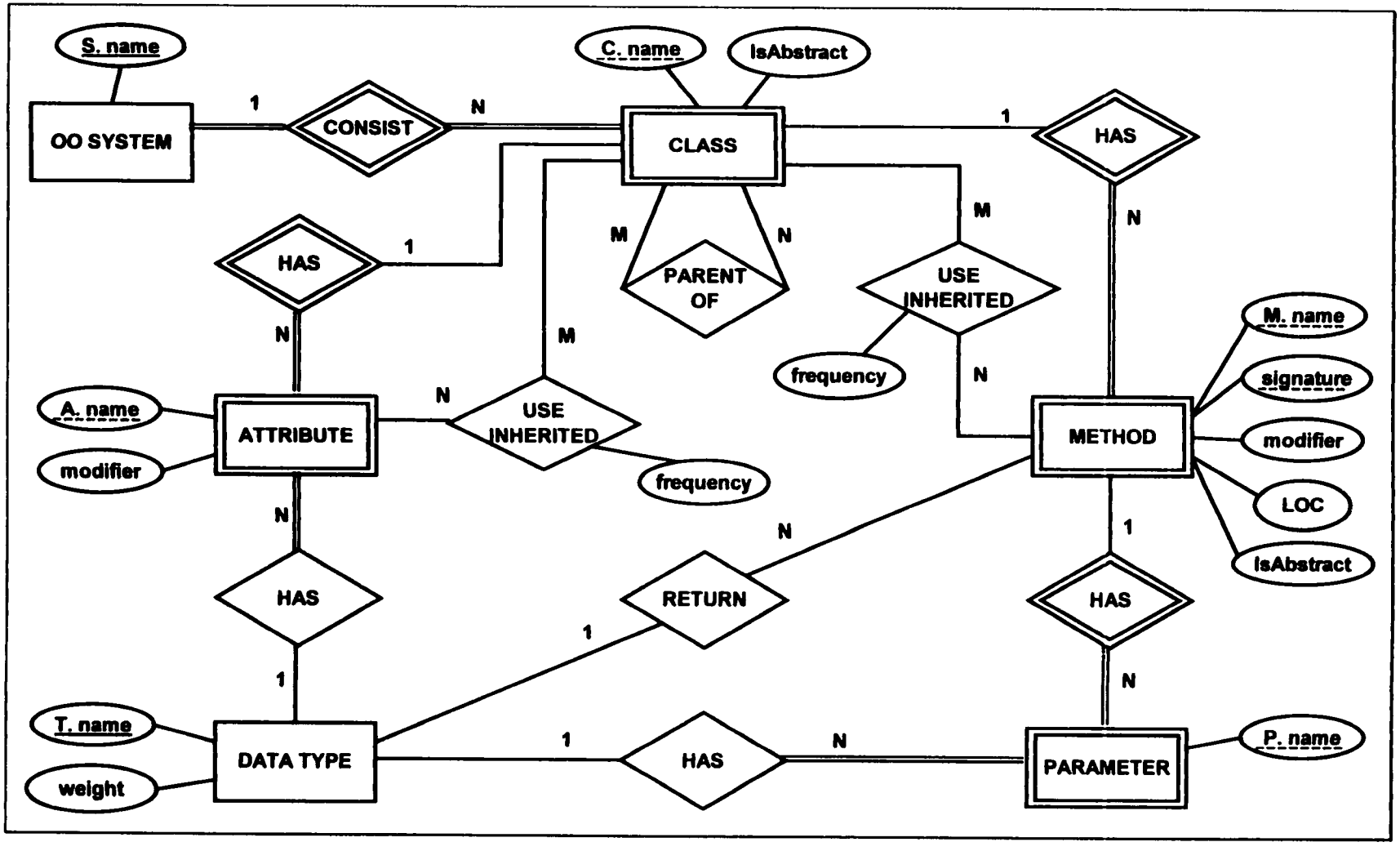


Figure 5.2: ER diagram of the tool.

5.2 Tool Implementation

The developed tool was implemented using *Javadoc in JDK 1.2*, *Microsoft Visual Basic 6.0 Enterprise Edition*, and *Microsoft Access 97*. Javadoc is a tool that parses the declarations and documentation comments in a set of Java source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields. Javadoc doclets can be used to customize Javadoc output. A doclet is a Java program written with the doclet API that specifies the content and format of the output to be generated by the Javadoc tool. The developed Java parser is a doclet with some modifications customized to parse any Java source file and produce a text file containing the inheritance coupling metrics data found in that file. The graphical user interface of the tool was designed using Microsoft Visual Basic. Microsoft Access is used as a database engine to connect to the central metrics repository.

5.3 Supported Metrics

The developed tool supports twenty-three object-oriented metrics varied in their applicability and purposes. Table 5.1 lists these metrics with a brief description of each one.

Metric	Description
Number of methods per class (NOM)	It counts the number of methods defined in a class.
Average number of methods per class (ANOM)	It calculates the average number of methods defined in all classes of the system.
Number of attributes per class (NOA)	It counts the number of attributes defined in a class.
Average number of attributes per class (ANOA)	It calculates the average number of attributes defined in all classes of the system.
Weight of methods per class (WOM)	It represents the complexity values (LOC) of all methods defined in a class.
Weight of attributes per class (WOA)	It represents the complexity values (weights) of all attributes defined in a class.
Depth of inheritance tree (DIT)	The length of the longest path from the class to the root class in the inheritance tree.
Average depth of inheritance tree (ADIT)	It is the average depth of inheritance tree for all classes in the system.
Number of children (NOC)	It counts the number of direct subclasses per class.
Average number of children (ANOC)	It calculates the average number of children for all classes in the system
Number of descendants (NOD)	It counts the number of direct and indirect subclasses per class.
Average number of descendants (ANOD)	It calculates the average number of descendants for all classes in the system
Individual class coupling (ICC)	It represents the inheritance coupling between a class and all classes in the system.
Class-to-class coupling (CTC)	It measures the inheritance coupling between any to classes in the system.
Overall system coupling (OSC)	It is the inheritance coupling of the whole OO system, i.e. average ICC of all classes in the system.
Coupling through inheritance (Henry and Li)	It is equal to the depth of inheritance tree (DIT) plus the number of children (NOC).
Attributes usability from a class (AUFC)	The percentage of inherited attributes from a class that are used by its subclasses to those that are not used.
Methods usability from a class (MUFC)	The percentage of inherited methods from a class that are used by its subclasses to those that are not used.
Attributes usability by a class (AUBC)	The percentage of inherited attributes by a class that are used by it to those that are not used.
Methods usability by a class (MUBC)	The percentage of inherited methods by a class that are used by it to those that are not used.
Usability-based Individual class coupling (UICC)	It represents the usability-based inheritance coupling between a class and all classes in the system.
Usability-based Class-to-class coupling (UCTC)	It measures the usability-based inheritance coupling between any to classes in the system.
Usability-based Overall system coupling (UOSC)	It is the usability-based inheritance coupling of the whole OO system, i.e. average UICC of all classes in the system.

Table 5.1: Metrics supported by the developed tool.

5.4 Features of the Tool

The following are some features and advantages of the developed tool:

- It simplifies the metrics data collection and ensures the accuracy and consistency of the collected data.
- It is not limited to particular language and support for new object-oriented languages and metrics can be added easily.
- It provides a platform for analyzing and comparing different object-oriented coupling metrics.
- It is easy to use through its graphical user interface.
- It provides chart view of the metrics results.
- It has class browser to navigate through system classes and their members.

5.5 Existing Tools

Few tools exist for automatically gathering, evaluating, and analyzing object-oriented metrics. In this section, three existing tools: Brooks and Buell's tool, TAC++, and OOMetDaGa are presented. A comparison between these tools and the developed tool is given in section 5.6.

5.5.1 Brooks and Buell's Tool

Brooks and Buell [Broo94] developed a tool to gather a subset of metrics

proposed by Chidamber and Kemerer [Chid94]. The tool was written in C++ and it only supports C++ software measurement. Brooks and Buell's tool [Broo94] is divided into two parts as shown in figure 5.3: a parsing and querying engine, and an object-oriented model class hierarchy. First, it will read C++ source code and gather meaningful data about classes in a database format. Second, the database can be queried to retrieve the various properties of a class or the system as a whole.

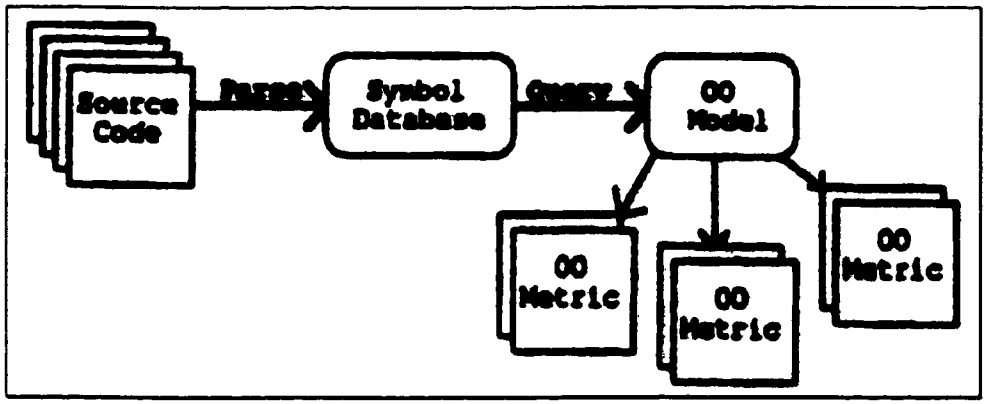


Figure 5.3: Brooks and Buell's tool structure [Broo94].

Brooks and Buell's tool [Broo94] supports two types of object-oriented metrics: class-level metrics and system-level metrics. The class-level metrics supported are:

- Depth of inheritance tree;

- Class coupling;
- Response for a class; and
- Number of children.

In addition, only one system-level metric is supported which is *number of class hierarchies*.

5.5.2 TAC++

TAC++ (Tool for Analyzing C++ code) [Bucc98, Fior98] is a research prototype suitable for studying the metric behavior and development process. It has been developed in several years and it can estimate a huge number of different metrics listed in table 5.2. The structure of TAC++ is shown in figure 5.4.

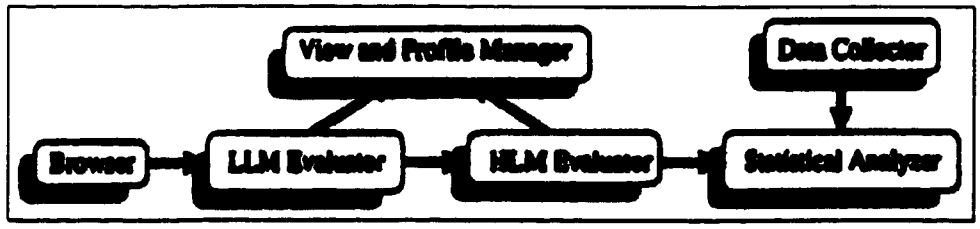


Figure 5.4: TAC++ structure [Bucc98].

Metric	Comment
CCm	Class Complexity/size
CCGI	Class CoGnitive Index
DIT	Deep Inheritance Tree
Ha	Halstead metric
HSCC	Class Complexity by Henderson-Sellers
LOC	number of Lines Of Code
Mc	McCabe ciclomatic Complexity
MCm	Method Complexity/size
NAL	Number of Attributes Locally defined of a class
NAM	Number of Attributes and Methods of a class
NSUP	Number of SUPerclasses of a class
Size2	Number of class attributes and methods
Tm	Total m-based functional complexity
TJCC	Class Complexity
WMC	Weighted Methods for Class

Table 5.2: Metrics supported by TAC++

TAC++ provides the following features [Bucc98, Fior98]:

- ***Navigating in the system classes***

  The class browser provides an integrated editor for navigating among system classes and their members.

- ***Evaluating low-level metrics (LLM)***

  TAC++ collects and analyzes low-level metrics (LLM) which are direct metrics such as Line-Of-Code, Halstead metric, number of attributes of a class, etc.

- ***Defining and evaluating high-level metrics (HLM)***

  High-level metrics (HLM) can be defined by the user on the basis of LLMs through visual editor.

- ***Defining and showing metric histograms and profiles***

  TAC++ provides two types of views for visualizing metrics results: profiles and histograms. A profile is a consumptive view to show the values of several metrics with respect to their mean, maximum and minimum values. Histograms are used to represent statistical data.

- ***Statically analyzing system for validating and tuning metrics***

  Validation of metrics can be performed using mathematical and statistical techniques. In TAC++, statistical analyzer with a graphical interface, based on Progress (a multi-linear regression tools), is used to validate metrics. This statistical analyzer is capable of estimating all the metric weights used in a metric if the corresponding real values are available.

- ***Collecting real data such as effort in person-hours, number of errors***

  Non-automatically measurable data such as effort are collected by filling forms or Java pages.

5.5.3 OOMetDaGa Environment

Object-Oriented Metrics Data Gathering (OOMetDaGa) environment [Heri98] provides an automatic support for object-oriented metrics data gathering. It acts as an integrating factor based on the central metrics repository that stores metric data on source code. Relations in OOMetDaGa environment are illustrated in figure 5.5. Basically there are three purposes of OOMetDaGa environment [Heri98]:

1. Metrics data collection;
2. Metrics data analysis; and
3. Dissemination of information via hypertext systems.

OOMetDaGa environment [Heri98] is not limited to particular language and support for new languages and metrics can be added. The collection engine of OOMetDaGa environment extracts metrics data using several ways. One way is through components that are integrated in development process, like classes in Visual Smalltalk or scripts in ParadigmPlus. Standalone tools and utility programs are other ways for metrics data extraction. For example, OOMetTool is used to define, validate, and evaluate C++ metrics, whereas PRO++ tool provides support to the software product auditing during acceptance phase. In addition, data can be added manually.

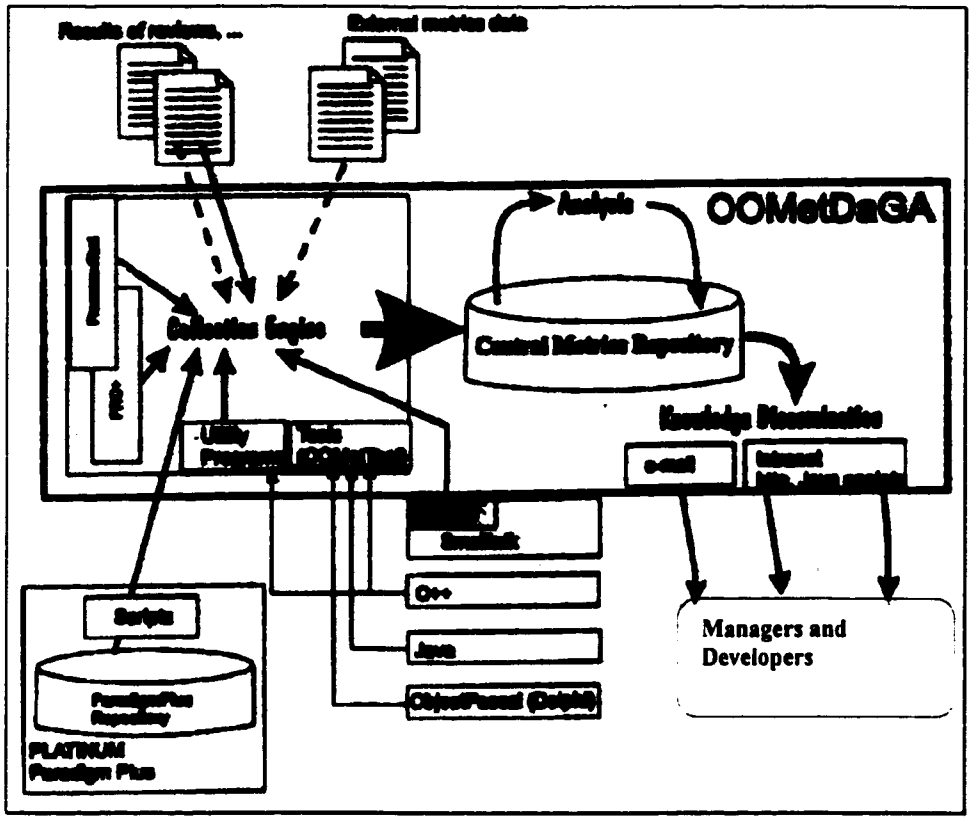


Figure 5.5: OOMetDaGa environment [Heri98].

5.6 Developed Tool vs. Existing Tools

Most of the existing tools support C++ software measurement, while the developed tool supports Java software measurement. Unlike the developed tool, existing tools do not support inter-class inheritance coupling metrics, i.e. measuring inheritance coupling between classes. Instead, they support simple inheritance coupling metrics, such as depth of inheritance tree (DIT), which are

also supported by the developed tool. The developed tool uses different concept, i.e. cover-coefficient concept presented in chapter 4, in calculating the coupling metrics. It separates the representation of the system from the calculation procedure of the coupling matrix. The advantage of such approach is that in order to evaluate the coupling between classes, efforts is only concentrated on how to define weights of attributes and methods in the corresponding classes [Almu98]. The developed tool, like some other existing tools, is not limited to particular language and support for new object-oriented languages and metrics can be added easily. Table 5.3 presents a comparison between the developed tool and the three tools described in this section 5.5.

Comparison Criteria	Tools			
	Brooks and Buell's Tool	**TAC++**	**OOMetDaGa Environment**	**Developed Tool**
Supported Languages	C++	C++	C++ , Java, ObjectPascal-Delphi, Smalltalk	Java
Number of supported metrics	5	15	?	23
Ability to support new languages and metrics	✓		✓	✓
Support for inheritance coupling metrics	✓	✓	✓	✓
Support for inter-class inheritance coupling metrics				✓
Separate system's representation from calculation procedure	✓		✓	✓
Framework used is applicable to other paradigms				✓

Table 5.3: Developed tool vs. existing tools.

CHAPTER 6

CASE STUDY

The developed tool was applied against six Java packages. Three of them are from Java Platform 1.2 API [JDK1.2] source code and the others are from Argo/UML [Argo] v0.7 source code. Measurement results of these packages are presented in sections 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6. Results analysis and observations are given in section 6.7. Appendix A shows the coupling matrices of these packages.

6.1 Package java.lang.ref

Package *java.lang.ref* is a six-class Java package from Java Platform 1.2 API source code. Table 6.1 lists the names of classes in this package, whereas figure 6.1 illustrates the class hierarchy. The measurement results summary is given in table 6.2. This includes the results of applying the general object-oriented metrics, i.e. NOM, NOA, DIT, NOC, and NOD, against this package. In addition, it shows the results of applying the inheritance coupling metrics:

the inheritance coupling metric proposed by AlMulla [Almu98], and coupling through inheritance metric proposed by Henry and Li [Li93].

Table 6.2 also displays the results of applying the usability metrics against this package. Two kinds of usability are measured: usability from a class and usability by a class. Usability from a class determines the percentage of inherited attributes and methods from each non-leaf class in the object-oriented system that are used by any descendant class(s) to those that are not used. By turn, usability by a class calculates the percentage of inherited attributes and methods by each non-root class in the object-oriented system that are used by it to those that are not used. The usability-based individual class coupling (UICC) values for each class in this package are given in table 6.2 as well.

Table A.1 shows the coupling matrix of this package, whereas table A.2 shows the usability-based coupling matrix of this package. Class-to-class (CTC) coupling values and usability-based class-to-class (UCTC) coupling values can be determined from these two tables respectively. Tables 6.3 and 6.4 give statistical information about the coupling matrix and the usability-based coupling matrix of this package respectively. They give the minimum, average, and maximum coupling values between each class in the package and

its ancestor, descendant, sibling, and other classes. This simplifies the understanding of the coupling matrices.

Class Number	Class Name
C1	java.lang.Object
C2	java.lang.ref.Reference
C3	java.lang.ref.ReferenceQueue
C4	java.lang.ref.PhantomReference
C5	java.lang.ref.SoftReference
C6	java.lang.ref.WeakReference

Table 6.1: Classes in package java.lang.ref.

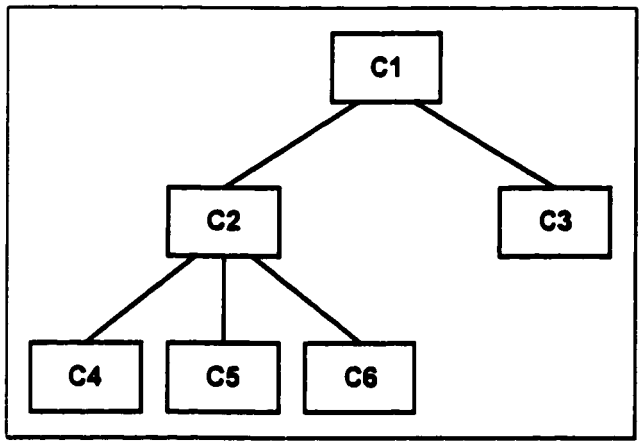


Figure 6.1: Class hierarchy of package java.lang.ref.

	General OO Metrics					Inheritance Coupling Metrics		Usability-Based Inheritance Coupling Metrics				
Class #	NOM	NOA	DIT	NOC	NOD	AlMulla (ICC)	Henry & Li	From		By		UICC
								A %	M %	A %	M %	
C1	13	0	0	2	5	0.5248	2		0			0
C2	6	5	1	3	3	0.6769	4	0	0		0	0
C3	6	4	1	0	0	0.3087	1				0	0
C4	2	0	2	0	0	0.7956	2			0	0	0
C5	3	2	2	0	0	0.7644	2			0	0	0
C6	2	0	2	0	0	0.7956	2			0	0	0
Avg	*5.33*	*1.83*	*1.33*	*0.83*	*1.33*	*0.6443*	*2.16*	*0*	*0*	*0*	*0*	*0*
					Var	*0.0376*					*Var*	*0*

Table 6.2: Measurement results summary of package java.lang.ref.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0.0702	0.1049	0.2438						
C2	0.2256	0.2256	0.2256	0.1392	0.1392	0.1392	0.0338	0.0338	0.0338			
C3	0.1545	0.1545	0.1545				0.0803	0.0803	0.0803	0.0246	0.0246	0.0246
C4	0.1419	0.2229	0.3039				0.1636	0.1636	0.1636	0.0226	0.0226	0.0226
C5	0.1364	0.2142	0.2920				0.1572	0.1572	0.1572	0.0217	0.0217	0.0217
C6	0.1419	0.2229	0.3039				0.1636	0.1636	0.1636	0.0226	0.0226	0.0226

Table 6.3: Coupling matrix analysis of package java.lang.ref.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0	0	0						
C2	0	0	0	0	0	0	0	0	0			
C3	0	0	0				0	0	0	0	0	0
C4	0	0	0				0	0	0	0	0	0
C5	0	0	0				0	0	0	0	0	0
C6	0	0	0				0	0	0	0	0	0

Table 6.4: Usability-based coupling matrix analysis of package java.lang.ref.

6.2 Package java.security.acl

Package *java.security.acl* is a six-class Java package from Java Platform 1.2 API source code. Table 6.5 lists the names of classes in this package, whereas figure 6.2 illustrates the class hierarchy. The measurement results summary is given in table 6.6. This includes the results of applying the general object-oriented metrics, i.e. NOM, NOA, DIT, NOC, and NOD, against this package. In addition, it shows the results of applying the inheritance coupling metrics. Table 6.6 also displays the results of applying the usability metrics against this package. The usability-based individual class coupling (UICC) values for each class in this package are given in table 6.6 as well. Table A.3 shows the coupling matrix of this package, whereas table A.4 shows the usability-based coupling matrix of this package. Class-to-class (CTC) coupling values and usability-based class-to-class (UCTC) coupling values can be determined from these two tables respectively. Tables 6.7 and 6.8 give statistical information about the coupling matrix and the usability-based coupling matrix of this package respectively. They give the minimum, average, and maximum coupling values between each class in the package and its ancestor, descendant, sibling, and other classes.

ClassNumber	ClassName
C1	java.lang.Object
C2	java.lang.Throwable
C3	java.lang.Exception
C4	java.security.acl.AclNotFoundException
C5	java.security.acl.LastOwnerException
C6	java.security.acl.NotOwnerException

Table 6.5: Classes in package java.security.acl.

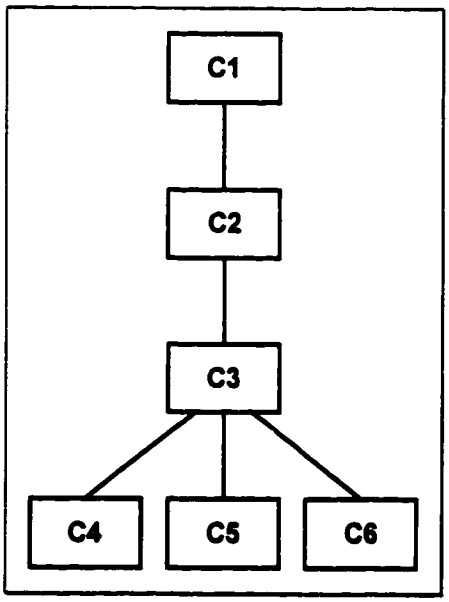


Figure 6.2: Class hierarchy of package java.security.acl.

	General OO Metrics					Inheritance Coupling Metrics		Usability-Based Inheritance Coupling Metrics				
Class #	NOM	NOA	DIT	NOC	NOD	AlMulla (ICC)	Henry & Li	From		By		UICC
								A %	M %	A %	M %	
C1	13	0	0	1	5	0.6157	1		9.1			0.4
C2	10	3	1	1	4	0.6711	2		0		9.1	0.6
C3	2	0	2	3	3	0.7460	5				0	0
C4	1	0	3	0	0	0.8884	3				0	0
C5	1	0	3	0	0	0.8884	3				0	0
C6	1	0	3	0	0	0.8884	3				0	0
Avg	*4.67*	*0.5*	*2*	*0.83*	*2*	*0.7830*	*2.83*	*0*	*3.03*	*0*	*1.82*	*0.1667*
					Var	*0.0150*					*Var*	*0.0707*

Table 6.6: Measurement results summary of package java.security.acl.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0.0578	0.1231	0.2619						
C2	0.2179	0.2179	0.2179	0.0752	0.1132	0.2274						
C3	0.1995	0.2510	0.3025	0.0813	0.0813	0.0813						
C4	0.1884	0.2407	0.2945				0.0831	0.0831	0.0831			
C5	0.1884	0.2407	0.2945				0.0831	0.0831	0.0831			
C6	0.1884	0.2407	0.2945				0.0831	0.0831	0.0831			

Table 6.7: Coupling matrix analysis of package java.security.acl.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0	0.08	0.4						
C2	0.6	0.6	0.6	0	0	0						
C3	0	0	0	0	0	0						
C4	0	0	0				0	0	0			
C5	0	0	0				0	0	0			
C6	0	0	0				0	0	0			

Table 6.8: Usability-based coupling matrix analysis of package java.security.acl.

6.3 Package uci.uml.critics.patterns

Package *uci.uml.critics.patterns* is a six-class Java package from Argo/UML v0.7 source code. Table 6.9 lists the names of classes in this package, whereas figure 6.3 illustrates the class hierarchy. The measurement results summary is given in table 6.10. This includes the results of applying the general object-oriented metrics, i.e. NOM, NOA, DIT, NOC, and NOD, against this package. In addition, it shows the results of applying the inheritance coupling metrics. Table 6.10 also displays the results of applying the usability metrics against this package. The usability-based individual class coupling (UICC) values for each class in this package are given in table 6.10 as well. Table A.5 shows the coupling matrix of this package, whereas table A.6 shows the usability-based coupling matrix of this package. Class-to-class (CTC) coupling values and usability-based class-to-class (UCTC) coupling values can be determined from these two tables respectively. Tables 6.11 and 6.12 give statistical information about the coupling matrix and the usability-based coupling matrix of this package respectively. They give the minimum, average, and maximum coupling values between each class in the package and its ancestor, descendant, sibling, and other classes.

Class Number	Class Name
C1	java.lang.Object
C2	uci.argo.kernel.Critic
C3	uci.uml.critics.patterns.PatternStereotypes
C4	uci.uml.critics.CrUML
C5	uci.uml.critics.patterns.CrConsiderSingleton
C6	uci.uml.critics.patterns.CrSingletonViolated

Table 6.9: Classes in package uci.uml.critics.patterns.

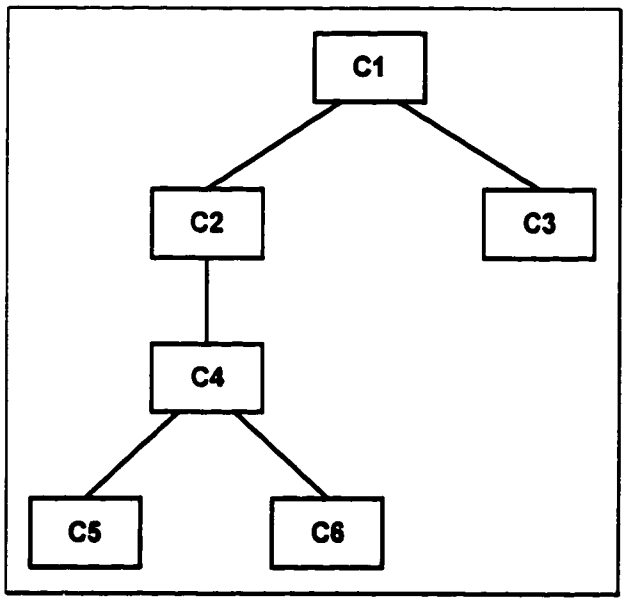


Figure 6.3: Class hierarchy of package uci.uml.critics.patterns.

	General OO Metrics					Inheritance Coupling Metrics		Usability-Based Inheritance Coupling Metrics				
Class #	NOM	NOA	DIT	NOC	NOD	AlMulla (ICC)	Henry & Li	From		By		UICC
								A %	M %	A %	M %	
C1	13	0	0	2	5	0.5190	2		18.2			0.4211
C2	65	32	1	1	3	0.6084	2	9.1	9.4		0	0.5439
C3	1	1	1	0	0	0.5447	1				0	0
C4	6	19	2	2	2	0.6911	4	0	40	4.5	4	0.6564
C5	3	0	3	0	0	0.7466	3			4.9	10	0.7789
C6	3	0	3	0	0	0.7526	3			2.4	7.5	0.7794
Avg	*15.17*	*8.67*	*1.67*	*0.83*	*1.67*	*0.6438*	*2.5*	*3.03*	*22.53*	*2.36*	*4.3*	*0.5299*
					Var	*0.0103*					*Var*	*0.0866*

Table 6.10: Measurement results summary of package uci.uml.critics.patterns.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0.0524	0.1038	0.2238						
C2	0.0446	0.0446	0.0446	0.1452	0.1863	0.2684	0.0049	0.0049	0.0049			
C3	0.2720	0.2720	0.2720				0.1281	0.1281	0.1281	0.0322	0.0482	0.0802
C4	0.0255	0.1370	0.2486	0.2071	0.2071	0.2071				0.0029	0.0029	0.0029
C5	0.0143	0.1726	0.3052				0.2272	0.2272	0.2272	0.0017	0.0017	0.0017
C6	0.0144	0.1740	0.3077				0.2291	0.2291	0.2291	0.0017	0.0017	0.0017

Table 6.11: Coupling matrix analysis of package uci.uml.critics.patterns.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0	0.0842	0.2632						
C2	0.1135	0.1135	0.1135	0.0658	0.1435	0.277	0	0	0			
C3	0	0	0				0	0	0	0	0	0
C4	0.0914	0.2316	0.3718	0.0703	0.0966	0.1229				0	0	0
C5	0	0.2343	0.3593				0.0758	0.0758	0.0758	0	0	0
C6	0	0.2233	0.373				0.1095	0.1095	0.1095	0	0	0

Table 6.12: Usability-based coupling matrix analysis of package uci.uml.critics.patterns.

6.4 Package uci.uml.util

Package *uci.uml.util* is a six-class Java package from Argo/UML v0.7 source code. Table 6.13 lists the names of classes in this package, whereas figure 6.4 illustrates the class hierarchy. The measurement results summary is given in table 6.14. This includes the results of applying the general object-oriented metrics, i.e. NOM, NOA, DIT, NOC, and NOD, against this package. In addition, it shows the results of applying the inheritance coupling metrics. Table 6.14 also displays the results of applying the usability metrics against this package. The usability-based individual class coupling (UICC) values for each class in this package are given in table 6.14 as well. Table A.7 shows the coupling matrix of this package, whereas table A.8 shows the usability-based coupling matrix of this package. Class-to-class (CTC) coupling values and usability-based class-to-class (UCTC) coupling values can be determined from these two tables respectively. Tables 6.15 and 6.16 give statistical information about the coupling matrix and the usability-based coupling matrix of this package respectively. They give the minimum, average, and maximum coupling values between each class in the package and its ancestor, descendant, sibling, and other classes.

Class Number	Class Name
C1	java.lang.Object
C2	uci.uml.util.GenAncestorClasses
C3	uci.uml.util.GenCompositeClasses
C4	uci.uml.util.GenDescendantClasses
C5	uci.uml.util.SuperclassGen
C6	uci.uml.util.UMLDescription

Table 6.13: Classes in package uci.uml.util.

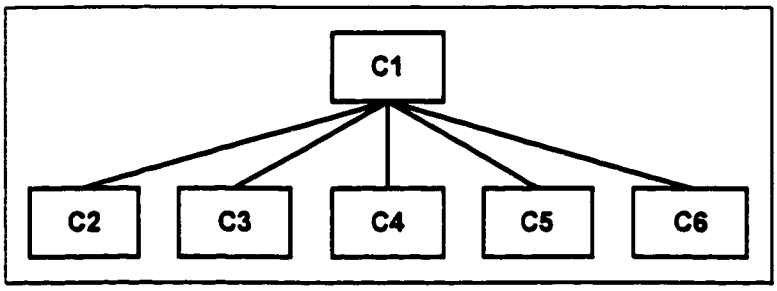


Figure 6.4: Class hierarchy of package uci.uml.util.

	General OO Metrics					Inheritance Coupling Metrics		Usability-Based Inheritance Coupling Metrics				
Class #	NOM	NOA	DIT	NOC	NOD	AlMulla (ICC)	Henry & Li	From		By		UICC
								A %	M %	A %	M %	
C1	13	0	0	5	5	0.4920	5		0			0
C2	3	1	1	0	0	0.4347	1				0	0
C3	2	1	1	0	0	0.4991	1				0	0
C4	3	1	1	0	0	0.4347	1				0	0
C5	2	0	1	0	0	0.6738	1				0	0
C6	3	10	1	0	0	0.1123	1				0	0
Avg	*4.33*	*2.17*	*0.83*	*0.83*	*0.83*	*0.4411*	*1.66*	*0*	*0*	*0*	*0*	*0*
					Var	*0.0337*					*Var*	*0.*

Table 6.14: Measurement results summary of package uci.uml.util.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0.0984	0.0984	0.0984						
C2	0.2381	0.2381	0.2381				0.0492	0.0492	0.0492			
C3	0.2733	0.2733	0.2733				0.0564	0.0564	0.0564			
C4	0.2381	0.2381	0.2381				0.0492	0.0492	0.0492			
C5	0.3690	0.3690	0.3690				0.0762	0.0762	0.0762			
C6	0.0615	0.0615	0.0615				0.0127	0.0127	0.0127			

Table 6.15: Coupling matrix analysis of package uci.uml.util.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0	0	0						
C2	0	0	0	0	0	0	0	0	0			
C3	0	0	0				0	0	0	0	0	0
C4	0	0	0				0	0	0	0	0	0
C5	0	0	0				0	0	0	0	0	0
C6	0	0	0				0	0	0	0	0	0

Table 6.16: Usability-based coupling matrix analysis of package uci.uml.util.

6.5 Package java.awt.event

Package *java.awt.event* is a twenty-four-class Java package from Java Platform 1.2 API source code. Table 6.17 lists the names of classes in this package, whereas figure 6.5 illustrates the class hierarchy. The measurement results summary is given in table 6.18. This includes the results of applying the general object-oriented metrics, i.e. NOM, NOA, DIT, NOC, and NOD, against this package. In addition, it shows the results of applying the inheritance coupling metrics. Table 6.18 also displays the results of applying the usability metrics against this package. The usability-based individual class coupling (UICC) values for each class in this package are given in table 6.18 as well. Tables A.9 and A.10 show the coupling matrix of this package, whereas tables A.11 and A.12 show the usability-based coupling matrix of this package. Class-to-class (CTC) coupling values and usability-based class-to-class (UCTC) coupling values can be determined from these tables. Tables 6.19 and 6.20 give statistical information about the coupling matrix and the usability-based coupling matrix of this package respectively. They give the minimum, average, and maximum coupling values between each class in the package and its ancestor, descendant, sibling, and other classes.

ClassNumber	ClassName
C1	java.lang.Object
C2	java.awt.event.ComponentAdapter
C3	java.awt.event.ContainerAdapter
C4	java.util.EventObject
C5	java.awt.event.FocusAdapter
C6	java.awt.event.KeyAdapter
C7	java.awt.event.MouseAdapter
C8	java.awt.event.MouseMotionAdapter
C9	java.awt.event.WindowAdapter
C10	java.awt.AWTEvent
C11	java.awt.event.ActionEvent
C12	java.awt.event.AdjustmentEvent
C13	java.awt.event.ComponentEvent
C14	java.awt.event.InputMethodEvent
C15	java.awt.event.InvocationEvent
C16	java.awt.event.ItemEvent
C17	java.awt.event.TextEvent
C18	java.awt.event.ContainerEvent
C19	java.awt.event.FocusEvent
C20	java.awt.event.InputEvent
C21	java.awt.event.PaintEvent
C22	java.awt.event.WindowEvent
C23	java.awt.event.KeyEvent
C24	java.awt.event.MouseEvent

Table 6.17: Classes in package java.awt.event.

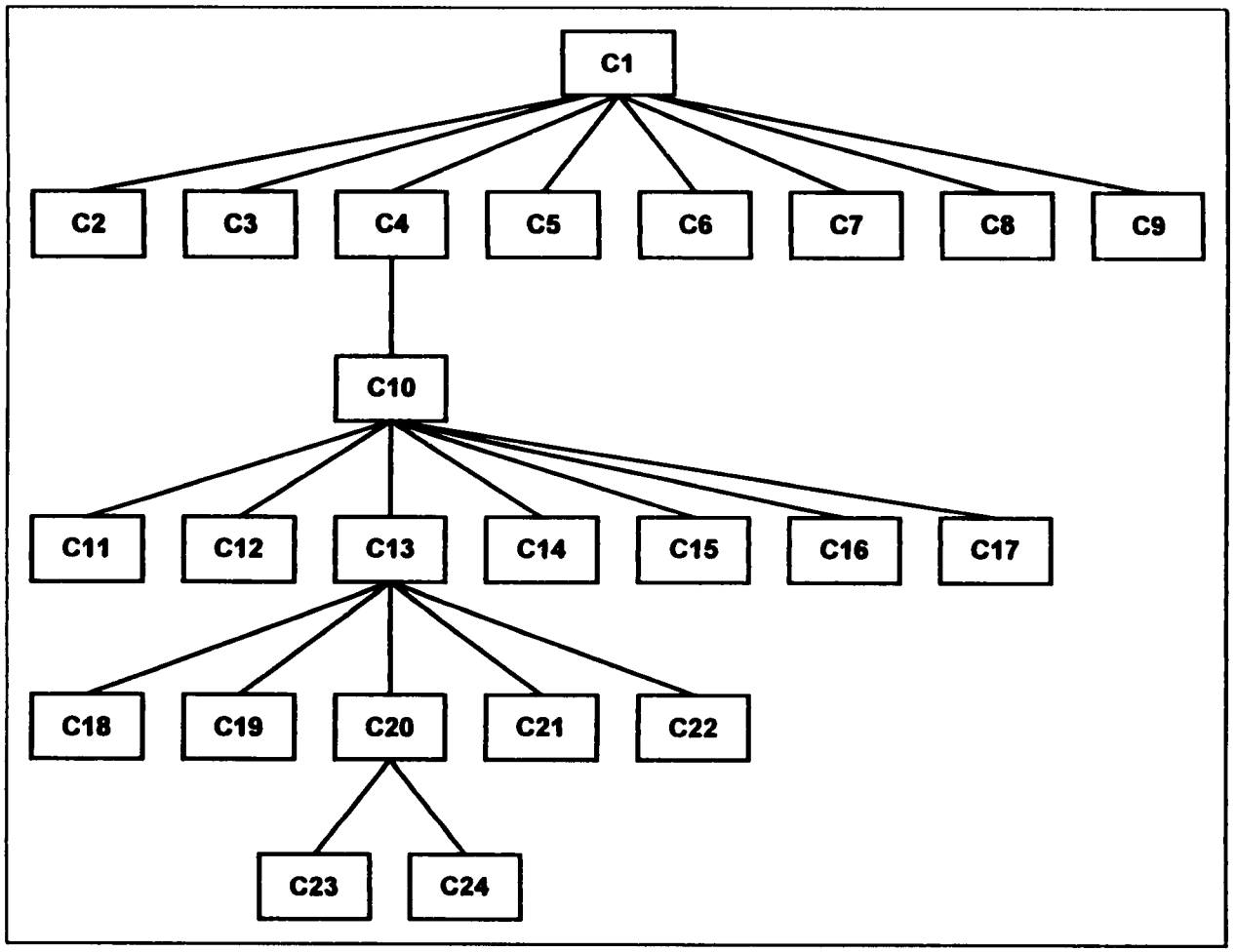


Figure 6.5: Class hierarchy of package java.awt.event.

	General OO Metrics					Inheritance Coupling Metrics		Usability-Based Inheritance Coupling Metrics				
								From		By		
Class #	NOM	NOA	DIT	NOC	NOD	AlMulla (ICC)	Henry & Li	A %	M %	A %	M %	UICC
C1	13	0	0	8	23	0.5894	8		9.1			0.48
C2	5	0	1	0	0	0.7447	1				0	0
C3	3	0	1	0	0	0.8274	1				0	0
C4	3	1	1	1	15	0.767	2	100	50		9.1	0.6851
C5	3	0	1	0	0	0.8274	1				0	0
C6	4	0	1	0	0	0.7839	1				0	0
C7	6	0	1	0	0	0.7092	1				0	0
C8	3	0	1	0	0	0.8274	1				0	0
C9	8	0	1	0	0	0.6476	1				0	0
C10	15	18	2	7	14	0.6558	9	12.5	0	100	15.4	0.6238
C11	5	10	3	0	0	0.7118	3			11.8	0	0.9609
C12	5	12	3	0	0	0.6676	3			11.8	0	0.9609
C13	3	7	3	5	7	0.8107	8	0	0	11.8	0	0.8451
C14	9	8	3	0	0	0.7677	3			17.6	0	0.9623
C15	6	7	3	0	0	0.7165	3			11.8	0	0.9609
C16	5	8	3	0	0	0.7362	3			11.8	0	0.9609
C17	2	4	3	0	0	0.8738	3			11.8	0	0.9609
C18	4	6	4	0	0	0.7996	4			8.7	0	0.9609
C19	4	6	4	0	0	0.8469	4			8.7	0	0.9609
C20	11	11	4	2	2	0.8222	6	18.2	11.1	13	0	0.7608
C21	4	6	4	0	0	0.7996	4			8.7	0	0.9609
C22	3	10	4	0	0	0.7662	4			8.7	0	0.9609
C23	12	192	5	0	0	0.2029	5			11.8	0	0.8766
C24	9	14	5	0	0	0.7047	5			11.8	2.9	0.8705
Average	*6.04*	*13.33*	*2.54*	*0.96*	*2.54*	*0.7336*	*3.50*	*26.14*	*14.04*	*11.3*	*1.19*	*0.6147*
					Var	*0.0178*					*Var*	*0.1772*

Table 6.18: Measurement results summary of package java.awt.event.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0.0065	0.0257	0.2007						
C2	0.3031	0.3031	0.3031				0.0053	0.0257	0.1484	0.0053	0.0174	0.1329
C3	0.3368	0.3368	0.3368				0.0059	0.0286	0.1649	0.0059	0.0194	0.1477
C4	0.3039	0.3039	0.3039	0.0097	0.0286	0.2089	0.0048	0.0048	0.0048			
C5	0.3368	0.3368	0.3368				0.0059	0.0286	0.1649	0.0059	0.0194	0.1477
C6	0.3191	0.3191	0.3191				0.0056	0.0271	0.1562	0.0056	0.0183	0.1399
C7	0.2887	0.2887	0.2887				0.0051	0.0246	0.1413	0.0051	0.0166	0.1266
C8	0.3368	0.3368	0.3368				0.0059	0.0286	0.1649	0.0059	0.0194	0.1477
C9	0.2636	0.2636	0.2636				0.0046	0.0224	0.1291	0.0046	0.0151	0.1156
C10	0.0828	0.0953	0.1079	0.0226	0.0324	0.1396				0.0017	0.0017	0.0017
C11	0.04	0.1047	0.2341				0.0233	0.0361	0.1	0.0007	0.0129	0.0361
C12	0.0375	0.0983	0.2196				0.0219	0.0339	0.0938	0.0007	0.0122	0.0339
C13	0.0661	0.1447	0.2833	0.0319	0.0356	0.0581	0.0196	0.0196	0.0196	0.0013	0.0013	0.0013
C14	0.0431	0.113	0.2525				0.0252	0.039	0.1078	0.0008	0.014	0.0389
C15	0.0402	0.1054	0.2357				0.0235	0.0364	0.1006	0.0007	0.013	0.0363
C16	0.0413	0.1083	0.2422				0.0241	0.0373	0.1034	0.0007	0.0133	0.0373
C17	0.0491	0.1286	0.2874				0.0286	0.0443	0.1227	0.0009	0.0159	0.0443
C18	0.0375	0.1118	0.2196				0.0327	0.0377	0.0527	0.0007	0.0134	0.0327
C19	0.0397	0.1184	0.2326				0.0347	0.04	0.0558	0.0007	0.0142	0.0347
C20	0.0413	0.107	0.2013	0.0923	0.0923	0.0923	0.026	0.026	0.026	0.0008	0.0081	0.0167
C21	0.0375	0.1118	0.2196				0.0327	0.0377	0.0527	0.0007	0.0134	0.0327
C22	0.0359	0.1072	0.2104				0.0314	0.0362	0.0505	0.0006	0.0129	0.0314
C23	0.0076	0.0256	0.0442				0.0212	0.0212	0.0212	0.0001	0.0031	0.0066
C24	0.0262	0.0889	0.1537				0.0736	0.0736	0.0736	0.0005	0.011	0.0229

Table 6.19: Coupling matrix analysis of package java.awt.event.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0	0.0209	0.2533						
C2	0	0	0				0	0	0	0	0	0
C3	0	0	0				0	0	0	0	0	0
C4	0.1607	0.1607	0.1607	0.0157	0.035	0.2833	0	0	0			
C5	0	0	0				0	0	0	0	0	0
C6	0	0	0				0	0	0	0	0	0
C7	0	0	0				0	0	0	0	0	0
C8	0	0	0				0	0	0	0	0	0
C9	0	0	0				0	0	0	0	0	0
C10	0.0977	0.1451	0.1925	0.0162	0.0238	0.1017				0	0	0
C11	0	0.1247	0.2259				0.0391	0.0504	0.107	0	0.0204	0.0504
C12	0	0.1247	0.2259				0.0391	0.0504	0.107	0	0.0204	0.0504
C13	0	0.151	0.3683	0.0278	0.0314	0.0532	0.0278	0.0286	0.0329	0	0	0
C14	0	0.1316	0.2573				0.0363	0.0498	0.1175	0	0.0192	0.051
C15	0	0.1247	0.2259				0.0391	0.0504	0.107	0	0.0204	0.0504
C16	0	0.1247	0.2259				0.0391	0.0504	0.107	0	0.0204	0.0504
C17	0	0.1247	0.2259				0.0391	0.0504	0.107	0	0.0204	0.0504
C18	0	0.1203	0.2259				0.0391	0.0419	0.0504	0	0.0209	0.0391
C19	0	0.1203	0.2259				0.0391	0.0419	0.0504	0	0.0209	0.0391
C20	0	0.0812	0.1786	0.1098	0.1208	0.1318	0.0193	0.0193	0.0193	0	0.009	0.021
C21	0	0.1203	0.2259				0.0391	0.0419	0.0504	0	0.0209	0.0391
C22	0	0.1203	0.2259				0.0391	0.0419	0.0504	0	0.0209	0.0391
C23	0	0.0998	0.1866				0.1234	0.1234	0.1234	0	0.0149	0.0254
C24	0	0.1022	0.2135				0.1176	0.1176	0.1176	0	0.0142	0.0242

Table 6.20: Usability-based coupling matrix analysis of package java.awt.event.

6.6 Package uci.uml.checklist

Package *uci.uml.checklist* is a seventeen-class Java package from Argo/UML v0.7 source code. Table 6.21 lists the names of classes in this package, whereas figure 6.6 illustrates the class hierarchy. The measurement results summary is given in table 6.22. This includes the results of applying the general object-oriented metrics, i.e. NOM, NOA, DIT, NOC, and NOD, against this package. In addition, it shows the results of applying the inheritance coupling metrics. Table 6.22 also displays the results of applying the usability metrics against this package. The usability-based individual class coupling (UICC) values for each class in this package are given in table 6.22 as well. Table A.13 shows the coupling matrix of this package, whereas table A.14 shows the usability-based coupling matrix of this package. Class-to-class (CTC) coupling values and usability-based class-to-class (UCTC) coupling values can be determined from these two tables respectively. Tables 6.23 and 6.24 give statistical information about the coupling matrix and the usability-based coupling matrix of this package respectively. They give the minimum, average, and maximum coupling values between each class in the package and its ancestor, descendant, sibling, and other classes.

ClassNumber	ClassName
C1	java.lang.Object
C2	uci.argo.checklist.CheckItem
C3	uci.argo.checklist.Checklist
C4	uci.uml.checklist.Init
C5	uci.uml.checklist.UMLCheckItem
C6	uci.uml.checklist.UMLChecklist
C7	uci.uml.checklist.ChActor
C8	uci.uml.checklist.ChAssociation
C9	uci.uml.checklist.ChAttribute
C10	uci.uml.checklist.ChClass
C11	uci.uml.checklist.ChInstance
C12	uci.uml.checklist.ChInterface
C13	uci.uml.checklist.ChLink
C14	uci.uml.checklist.ChOperation
C15	uci.uml.checklist.ChState
C16	uci.uml.checklist.ChTransition
C17	uci.uml.checklist.ChUseCase

Table 6.21: Classes in package uci.uml.checklist.

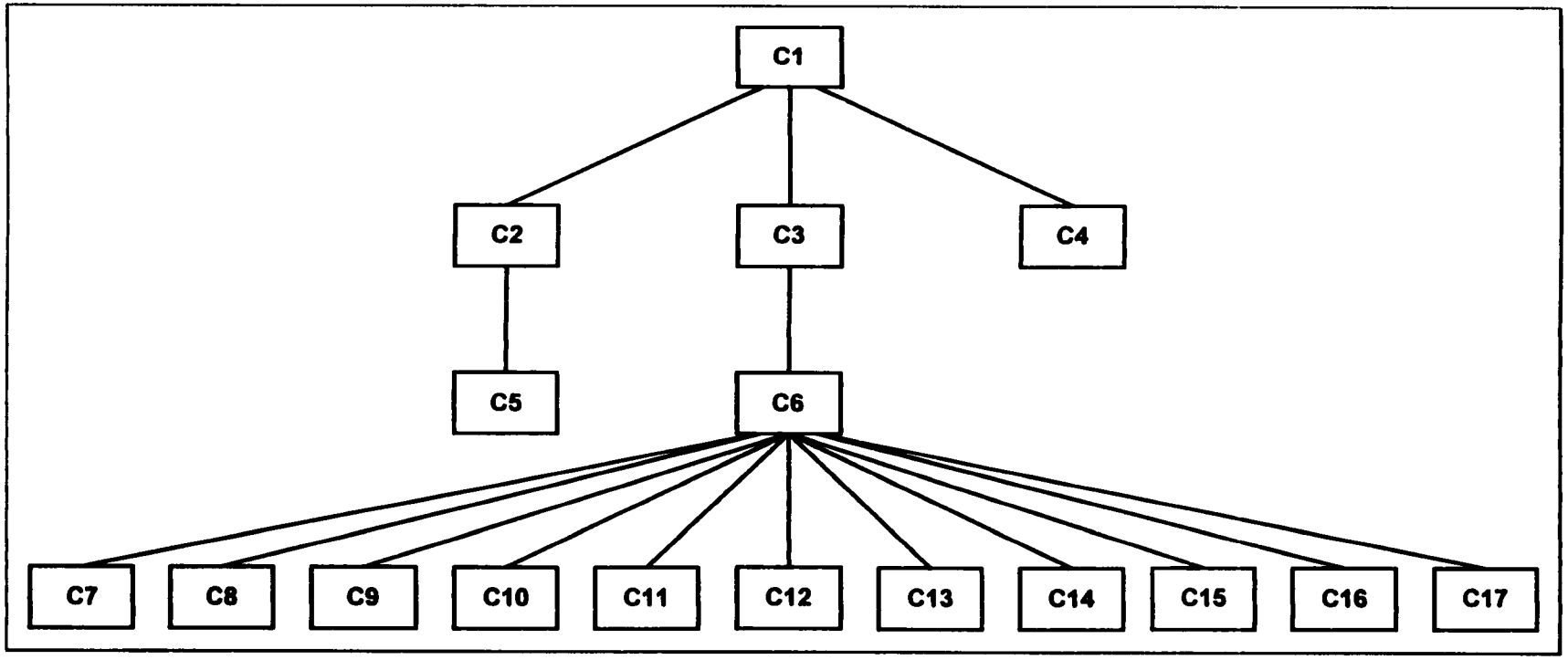


Figure 6.6: Class hierarchy of package uci.uml.checklist.

	General OO Metrics					Inheritance Coupling Metrics		Usability-Based Inheritance Coupling Metrics				
Class #	NOM	NOA	DIT	NOC	NOD	AlMulla (ICC)	Henry & Li	From		By		UICC
								A %	M %	A %	M %	
C1	13	0	0	3	16	0.6212	3		9.1			0.359
C2	14	4	1	1	1	0.5579	2	0	0		0	0
C3	12	2	1	1	12	0.7291	2	50	30		9.1	0.583
C4	1	11	1	0	0	0.0931	1				0	0
C5	3	0	2	0	0	0.6015	2			0	0	0
C6	2	0	2	11	11	0.7471	13		100	50	0	0.5948
C7	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C8	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C9	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C10	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C11	1	0	3	0	0	0.8644	3			0	18.2	0.9689
C12	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C13	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C14	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C15	1	0	3	0	0	0.8816	3			0	18.2	0.9689
C16	1	0	3	0	0	0.8996	3			0	18.2	0.9689
C17	1	0	3	0	0	0.8816	3			0	18.2	0.9689
Average	*3.29*	*1*	*2.35*	*0.94*	*2.35*	*0.7676*	*3.29*	*12.5*	*34.78*	*3.12*	*13.08*	*0.7173*
					Var	*0.0434*					*Var*	*0.1501*

Table 6.22: Measurement results summary of package uci.uml.checklist.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0.0152	0.0389	0.2121						
C2	0.0947	0.0947	0.0947	0.3193	0.3193	0.3193	0.0038	0.0284	0.053	0.0038	0.0073	0.0455
C3	0.1725	0.1725	0.1725	0.0269	0.0441	0.2334	0.0069	0.0104	0.0138	0.0069	0.0069	0.0069
C4	0.0358	0.0358	0.0358				0.0029	0.0114	0.02	0.0014	0.0026	0.0172
C5	0.0649	0.2514	0.4378							0.0026	0.0071	0.0364
C6	0.159	0.205	0.2511	0.0283	0.0283	0.0283				0.0064	0.0085	0.0127
C7	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C8	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C9	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C10	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C11	0.0891	0.158	0.1944				0.0376	0.0376	0.0376	0.0036	0.0048	0.0071
C12	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C13	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C14	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C15	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073
C16	0.0928	0.1645	0.2024				0.0391	0.0391	0.0391	0.0037	0.0049	0.0074
C17	0.0909	0.1612	0.1983				0.0384	0.0384	0.0384	0.0036	0.0048	0.0073

Table 6.23: Coupling matrix analysis of package uci.uml.checklist.

Class Name	Ancestor Classes			Descendant Classes			Sibling Classes			Other Classes		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
C1				0	0.0224	0.359						
C2	0	0	0	0	0	0	0	0	0	0	0	0
C3	0.1136	0.1136	0.1136	0.0144	0.0392	0.3114	0	0	0	0	0	0
C4	0	0	0				0	0	0	0	0	0
C5	0	0	0							0	0	0
C6	0	0.1782	0.3565	0.0217	0.0217	0.0217				0	0	0
C7	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C8	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C9	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C10	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C11	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C12	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C13	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C14	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C15	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C16	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0
C17	0	0.2192	0.3737				0.0311	0.0311	0.0311	0	0	0

Table 6.24: Usability-based coupling matrix analysis of package uci.uml.checklist.

6.7 Results Analysis and Observations

The following points can be observed from the measurement results of the six packages presented in this chapter.

1. A class is more coupled to its closer children (subclasses) than far ones. For example, in package java.security.acl, $C_{2,3}$ is greater than $C_{2,4}$, $C_{2,5}$, and $C_{2,6}$ and so on.

2. The deeper we go for the same class, the coupling values get decreasing and vice versa. For instance, in package uci.uml.critics.patterns, $C_{1,2} > C_{1,4} > C_{1,5}$ and so on.

3. ICC values of sibling leaf classes are equal if these classes have the same size in term of the number of attributes and methods they have and their weights. For instance, ICC of C4, C5, and C6 are equal to 0.8884 in package java.security.acl. However, in package java.lang.ref, ICC of C5 is smaller than ICC of C4 and C6 because C5 has relatively larger size than C4 and C6.

4. Coupling values between sibling leaf classes and their parent class are equal if these classes have the same size in term of the number of attributes and

methods they have and their weights. For instance, in package java.security.acl, $C_{4,3} = C_{5,3} = C_{6,3} = 0.2393$. However, in package java.awt.event, $C_{23,20}$ is smaller than $C_{24,20}$ because C23 has relatively larger size than C24.

5. Coupling values between sibling leaf classes are equal if these classes have the same size in term of the number of attributes and methods they have and their weights. For example, in package java.security.acl, $C_{4,5} = C_{4,6} = C_{5,4} = C_{5,6} = C_{6,4} = C_{6,5} = 0.0831$.

6. UICC values of the leaf classes whose direct super-class is java.lang.Object class are most likely to be zero because java.lang.Object class is the root class of any Java class hierarchy.

7. Many classes have the same coupling values when Henry and Li metric was applied because this metric considers all classes to have equal complexity and ignores the internal complexity of the classes. However, this is not the case when AlMulla and Usability-based inheritance coupling metrics were applied because these two metrics consider classes complexity/size in coupling calculation.

8. OSC value of a package in the usability-based inheritance coupling metric is less than or equal to OSC value of the inheritance coupling metric proposed by AlMulla. This is because each entry in the definition matrix of the usability-based inheritance coupling metric is equal to either the corresponding entry in the definition matrix of AlMulla metric or zero.

9. The coupling matrices as well as the usability-based coupling matrices of the six case studies were tested against the following questions:
   a. *Are all classes more coupled to their ancestor and descendant classes than to other classes?*
   b. *Are all classes more coupled to their sibling classes than to non-ancestor and non-descendant classes?*
   c. *Are all classes more coupled to their ancestor classes than to their descendant classes?*

   The test results are given in table 6.25. As shown in this table, some packages do not satisfy some of these questions. A future research is needed to determine whether good class hierarchies should satisfy these questions or not. Furthermore, if a system doesn't satisfy any of these questions does this indicate a problem in its class hierarchy or not.

	AlMulla Coupling Matrix						Usability-based Coupling Matrix					
	Package java.lang.ref	Package java.security.acl	Package uci.uml.critics.patterns	Package uci.uml.util	Package java.awt.event	Package uci.uml.checklist	Package java.lang.ref	Package java.security.acl	Package uci.uml.critics.patterns	Package uci.uml.util	Package java.awt.event	Package uci.uml.checklist
Are all classes more coupled to their ancestor and descendant classes than to other classes?	✗	✓	✗	✓	✗	✓	✓	✓	✗	✓	✗	✗
Are all classes more coupled to their sibling classes than to non-ancestor and non-descendant classes?	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✓
Are all classes more coupled to their ancestor classes than to their descendant classes?	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✗	✓

Table 6.25: Test results of the coupling matrices of all case studies.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Software metrics play an important role in evaluating and improving development processes and software products. According to DeMarco's principle: "you cannot control what you cannot measure". Object-oriented metrics can be classified into three levels of applicability: method-level metrics, class-level metrics, and system-level metrics. Coupling is one of the design quality attributes as far as complexity and maintainability of a design are concerned. It is defined as a measure of the strength of component interconnections. In object-oriented design, three types of class coupling exist: inheritance coupling, interaction coupling, and component coupling.

A tool for measuring inheritance coupling in Java programs was developed. It consists of three main components: parsing engines, central metrics repository, and query engines. Three object-oriented metrics tools: Brooks and Buell's tool, TAC++, and OOMetDaGa environment are presented in this thesis and compared to the developed tool. The comparison shows that

the developed tool has some features that are not available in the other tools. It also indicates that the uniqueness of the developed tool relies on the framework it uses to calculate the coupling.

7.1 Major Contributions

The major contributions of this thesis can be summarized as follows.

1. A tool for measuring inheritance coupling in object-oriented systems was developed. This tool satisfies the metrics requirements by automatically gathering, evaluating, and analyzing the supported metrics. In addition, it simplifies the inheritance coupling metrics data collection and ensures the accuracy and consistency of the collected data. This tool also provides a platform for analyzing and comparing different inheritance coupling metrics. Although the tool is currently supporting Java programs, it is not limited to particular language and support for new object-oriented languages and metrics can be added easily. As parts of this tool:
   - A Java parser that can parse any Java source code to collect inheritance coupling metrics data was developed.
   - An abstract (language independent) representation of object-oriented systems from inheritance point of view was modeled as ER model.

2. The inheritance coupling metric, proposed by AlMulla [Almu98], was enhanced by considering the usability of inherited attributes and methods by inheriting classes in coupling calculation.
3. Two usability metrics were proposed: one measures the usage percentage of inherited elements from a class, and the other measures the usage percentage of inherited elements by inheriting class.
4. Inheritance coupling was redefined in more appropriate way that takes into account indirect inheritance relationships that may exist between classes.
5. Some inheritance guidelines were set to assist object-oriented designers in building good inheritance hierarchies.

7.2 Future Work

The following research activities are candidate areas for future work:

- Setting guidelines for interpreting the coupling matrix values, and suggesting some numeric thresholds.
- Developing parsers for other object-oriented languages such as C++ and SmallTalk and designs such as UML.
- Tuning the weighing schemes used to assign weights to the factors that influence the inheritance coupling.

- Validating the measurement results and the framework using mathematical and experimental techniques.
- Studying the relationship between the inheritance coupling and other types of coupling in object-oriented design.

APPENDIX A

(Coupling matrices of the case studies)

	C1	C2	C3	C4	C5	C6
C1	0.4752	0.2438	0.0702	0.0702	0.0702	0.0702
C2	0.2256	0.3231	0.0338	0.1392	0.1392	0.1392
C3	0.1545	0.0803	0.6913	0.0246	0.0246	0.0246
C4	0.1419	0.3039	0.0226	0.2044	0.1636	0.1636
C5	0.1364	0.2920	0.0217	0.1572	0.2356	0.1572
C6	0.1419	0.3039	0.0226	0.1636	0.1636	0.2044

Table A.1: Coupling matrix of package java.lang.ref.

	C1	C2	C3	C4	C5	C6
C1	1	0	0	0	0	0
C2	0	1	0	0	0	0
C3	0	0	1	0	0	0
C4	0	0	0	1	0	0
C5	0	0	0	0	1	0
C6	0	0	0	0	0	1

Table A.2: Usability-based coupling matrix of package java.lang.ref.

	C1	C2	C3	C4	C5	C6
C1	0.3843	0.2619	0.1803	0.0578	0.0578	0.0578
C2	0.2179	0.3289	0.2274	0.0752	0.0752	0.0752
C3	0.1995	0.3025	0.2540	0.0813	0.0813	0.0813
C4	0.1884	0.2945	0.2393	0.1116	0.0831	0.0831
C5	0.1884	0.2945	0.2393	0.0831	0.1116	0.0831
C6	0.1884	0.2945	0.2393	0.0831	0.0831	0.1116

Table A.3: Coupling matrix of package java.security.acl.

	C1	C2	C3	C4	C5	C6
C1	0.6	0.4	0	0	0	0
C2	0.6	0.4	0	0	0	0
C3	0	0	1	0	0	0
C4	0	0	0	1	0	0
C5	0	0	0	0	1	0
C6	0	0	0	0	0	1

Table A.4: Usability-based coupling matrix of package java.security.acl.

	C1	C2	C3	C4	C5	C6
C1	0.4810	0.2238	0.0524	0.1381	0.0524	0.0524
C2	0.0446	0.3916	0.0049	0.2684	0.1452	0.1452
C3	0.2720	0.1281	0.4553	0.0802	0.0322	0.0322
C4	0.0255	0.2486	0.0029	0.3089	0.2071	0.2071
C5	0.0143	0.1982	0.0017	0.3052	0.2534	0.2272
C6	0.0144	0.1998	0.0017	0.3077	0.2291	0.2474

Table A.5: Coupling matrix of package uci.uml.critics.patterns.

	C1	C2	C3	C4	C5	C6
C1	0.5789	0.2632	0	0.1579	0	0
C2	0.1135	0.4561	0	0.277	0.0876	0.0658
C3	0	0	1	0	0	0
C4	0.0914	0.3718	0	0.3436	0.1229	0.0703
C5	0	0.3437	0	0.3593	0.2211	0.0758
C6	0	0.373	0	0.2968	0.1095	0.2206

Table A.6: Usability-based coupling matrix of package uci.uml.critics.patterns.

	C1	C2	C3	C4	C5	C6
C1	0.5080	0.0984	0.0984	0.0984	0.0984	0.0984
C2	0.2381	0.5653	0.0492	0.0492	0.0492	0.0492
C3	0.2733	0.0564	0.5009	0.0564	0.0564	0.0564
C4	0.2381	0.0492	0.0492	0.5653	0.0492	0.0492
C5	0.3690	0.0762	0.0762	0.0762	0.3262	0.0762
C6	0.0615	0.0127	0.0127	0.0127	0.0127	0.8877

Table A.7: Coupling matrix of package uci.uml.util.

	C1	C2	C3	C4	C5	C6
C1	1	0	0	0	0	0
C2	0	1	0	0	0	0
C3	0	0	1	0	0	0
C4	0	0	0	1	0	0
C5	0	0	0	0	1	0
C6	0	0	0	0	0	1

Table A.8: Usability-based coupling matrix of package uci.uml.util.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
C1	0.4106	0.0065	0.0065	0.2007	0.0065	0.0065	0.0065	0.0065	0.0065	0.1797	0.0065	0.0065
C2	0.3031	0.2553	0.0053	0.1484	0.0053	0.0053	0.0053	0.0053	0.0053	0.1329	0.0053	0.0053
C3	0.3368	0.0059	0.1726	0.1649	0.0059	0.0059	0.0059	0.0059	0.0059	0.1477	0.0059	0.0059
C4	0.3039	0.0048	0.0048	0.233	0.0048	0.0048	0.0048	0.0048	0.0048	0.2089	0.0097	0.0097
C5	0.3368	0.0059	0.0059	0.1649	0.1726	0.0059	0.0059	0.0059	0.0059	0.1477	0.0059	0.0059
C6	0.3191	0.0056	0.0056	0.1562	0.0056	0.2161	0.0056	0.0056	0.0056	0.1399	0.0056	0.0056
C7	0.2887	0.0051	0.0051	0.1413	0.0051	0.0051	0.2908	0.0051	0.0051	0.1266	0.0051	0.0051
C8	0.3368	0.0059	0.0059	0.1649	0.0059	0.0059	0.0059	0.1726	0.0059	0.1477	0.0059	0.0059
C9	0.2636	0.0046	0.0046	0.1291	0.0046	0.0046	0.0046	0.0046	0.3524	0.1156	0.0046	0.0046
C10	0.1079	0.0017	0.0017	0.0828	0.0017	0.0017	0.0017	0.0017	0.0017	0.3442	0.0226	0.0226
C11	0.0401	0.0007	0.0007	0.04	0.0007	0.0007	0.0007	0.0007	0.0007	0.2341	0.2882	0.0233
C12	0.0377	0.0007	0.0007	0.0375	0.0007	0.0007	0.0007	0.0007	0.0007	0.2196	0.0219	0.3324
C13	0.0847	0.0013	0.0013	0.0661	0.0013	0.0013	0.0013	0.0013	0.0013	0.2833	0.0196	0.0196
C14	0.0433	0.0008	0.0008	0.0431	0.0008	0.0008	0.0008	0.0008	0.0008	0.2525	0.0252	0.0252
C15	0.0404	0.0007	0.0007	0.0402	0.0007	0.0007	0.0007	0.0007	0.0007	0.2357	0.0235	0.0235
C16	0.0415	0.0007	0.0007	0.0413	0.0007	0.0007	0.0007	0.0007	0.0007	0.2422	0.0241	0.0241
C17	0.0493	0.0009	0.0009	0.0491	0.0009	0.0009	0.0009	0.0009	0.0009	0.2874	0.0286	0.0286
C18	0.0377	0.0007	0.0007	0.0375	0.0007	0.0007	0.0007	0.0007	0.0007	0.2196	0.0219	0.0219
C19	0.0399	0.0007	0.0007	0.0397	0.0007	0.0007	0.0007	0.0007	0.0007	0.2326	0.0232	0.0232
C20	0.0487	0.0008	0.0008	0.0413	0.0008	0.0008	0.0008	0.0008	0.0008	0.2013	0.0167	0.0167
C21	0.0377	0.0007	0.0007	0.0375	0.0007	0.0007	0.0007	0.0007	0.0007	0.2196	0.0219	0.0219
C22	0.0361	0.0006	0.0006	0.0359	0.0006	0.0006	0.0006	0.0006	0.0006	0.2104	0.021	0.021
C23	0.0076	0.0001	0.0001	0.0076	0.0001	0.0001	0.0001	0.0001	0.0001	0.0442	0.0044	0.0044
C24	0.0264	0.0005	0.0005	0.0262	0.0005	0.0005	0.0005	0.0005	0.0005	0.1537	0.0153	0.0153

Table A.9: Coupling matrix I of package java.awt.event.

	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24
C1	0.0694	0.0065	0.0065	0.0065	0.0065	0.0065	0.0065	0.017	0.0065	0.0065	0.0065	0.0065
C2	0.0517	0.0053	0.0053	0.0053	0.0053	0.0053	0.0053	0.013	0.0053	0.0053	0.0053	0.0053
C3	0.0575	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0145	0.0059	0.0059	0.0059	0.0059
C4	0.0821	0.0097	0.0097	0.0097	0.0097	0.0097	0.0097	0.0218	0.0097	0.0097	0.0097	0.0097
C5	0.0575	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0145	0.0059	0.0059	0.0059	0.0059
C6	0.0544	0.0056	0.0056	0.0056	0.0056	0.0056	0.0056	0.0137	0.0056	0.0056	0.0056	0.0056
C7	0.0493	0.0051	0.0051	0.0051	0.0051	0.0051	0.0051	0.0124	0.0051	0.0051	0.0051	0.0051
C8	0.0575	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059	0.0145	0.0059	0.0059	0.0059	0.0059
C9	0.045	0.0046	0.0046	0.0046	0.0046	0.0046	0.0046	0.0113	0.0046	0.0046	0.0046	0.0046
C10	0.1396	0.0226	0.0226	0.0226	0.0226	0.0226	0.0226	0.0421	0.0226	0.0226	0.0226	0.0226
C11	0.1	0.0233	0.0233	0.0233	0.0233	0.0233	0.0233	0.0361	0.0233	0.0233	0.0233	0.0233
C12	0.0938	0.0219	0.0219	0.0219	0.0219	0.0219	0.0219	0.0339	0.0219	0.0219	0.0219	0.0219
C13	0.1893	0.0196	0.0196	0.0196	0.0196	0.0319	0.0319	0.0581	0.0319	0.0319	0.0319	0.0319
C14	0.1078	0.2323	0.0252	0.0252	0.0252	0.0252	0.0252	0.0389	0.0252	0.0252	0.0252	0.0252
C15	0.1006	0.0235	0.2835	0.0235	0.0235	0.0235	0.0235	0.0363	0.0235	0.0235	0.0235	0.0235
C16	0.1034	0.0241	0.0241	0.2638	0.0241	0.0241	0.0241	0.0373	0.0241	0.0241	0.0241	0.0241
C17	0.1227	0.0286	0.0286	0.0286	0.1262	0.0286	0.0286	0.0443	0.0286	0.0286	0.0286	0.0286
C18	0.1526	0.0219	0.0219	0.0219	0.0219	0.2004	0.0327	0.0527	0.0327	0.0327	0.0327	0.0327
C19	0.1616	0.0232	0.0232	0.0232	0.0232	0.0347	0.1531	0.0558	0.0347	0.0347	0.0347	0.0347
C20	0.1369	0.0167	0.0167	0.0167	0.0167	0.026	0.026	0.1778	0.026	0.026	0.0923	0.0923
C21	0.1526	0.0219	0.0219	0.0219	0.0219	0.0327	0.0327	0.0527	0.2004	0.0327	0.0327	0.0327
C22	0.1462	0.021	0.021	0.021	0.021	0.0314	0.0314	0.0505	0.0314	0.2338	0.0314	0.0314
C23	0.0307	0.0044	0.0044	0.0044	0.0044	0.0066	0.0066	0.0378	0.0066	0.0066	0.7971	0.0212
C24	0.1068	0.0153	0.0153	0.0153	0.0153	0.0229	0.0229	0.1313	0.0229	0.0229	0.0736	0.2953

Table A.10: Coupling matrix II of package java.awt.event.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
C1	0.52	0	0	0.2533	0	0	0	0	0	0.2267	0	0
C2	0	1	0	0	0	0	0	0	0	0	0	0
C3	0	0	1	0	0	0	0	0	0	0	0	0
C4	0.1607	0	0	0.3149	0	0	0	0	0	0.2833	0.0157	0.0157
C5	0	0	0	0	1	0	0	0	0	0	0	0
C6	0	0	0	0	0	1	0	0	0	0	0	0
C7	0	0	0	0	0	0	1	0	0	0	0	0
C8	0	0	0	0	0	0	0	1	0	0	0	0
C9	0	0	0	0	0	0	0	0	1	0	0	0
C10	0.0977	0	0	0.1925	0	0	0	0	0	0.3762	0.0162	0.0162
C11	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C12	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C13	0	0	0	0.0848	0	0	0	0	0	0.3683	0.0278	0.0278
C14	0	0	0	0.1376	0	0	0	0	0	0.2573	0.0363	0.0363
C15	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C16	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C17	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C18	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C19	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C20	0	0	0	0.068	0	0	0	0	0	0.1786	0.0193	0.0193
C21	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C22	0	0	0	0.1482	0	0	0	0	0	0.2259	0.0391	0.0391
C23	0	0	0	0.0963	0	0	0	0	0	0.1468	0.0254	0.0254
C24	0	0	0	0.0917	0	0	0	0	0	0.1398	0.0242	0.0242

Table A.11: Usability-based coupling matrix I of package java.awt.event.

	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24
C1	0	0	0	0	0	0	0	0	0	0	0	0
C2	0	0	0	0	0	0	0	0	0	0	0	0
C3	0	0	0	0	0	0	0	0	0	0	0	0
C4	0.0345	0.0157	0.0157	0.0157	0.0157	0.0157	0.0157	0.0188	0.0157	0.0157	0.0157	0.0157
C5	0	0	0	0	0	0	0	0	0	0	0	0
C6	0	0	0	0	0	0	0	0	0	0	0	0
C7	0	0	0	0	0	0	0	0	0	0	0	0
C8	0	0	0	0	0	0	0	0	0	0	0	0
C9	0	0	0	0	0	0	0	0	0	0	0	0
C10	0.1017	0.0199	0.0162	0.0162	0.0162	0.0162	0.0162	0.0335	0.0162	0.0162	0.0162	0.0162
C11	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C12	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C13	0.1549	0.0329	0.0278	0.0278	0.0278	0.0278	0.0278	0.0532	0.0278	0.0278	0.0278	0.0278
C14	0.1175	0.0377	0.0363	0.0363	0.0363	0.0363	0.0363	0.051	0.0363	0.0363	0.0363	0.0363
C15	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C16	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C17	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C18	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C19	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C20	0.0783	0.021	0.0193	0.0193	0.0193	0.0193	0.0193	0.2392	0.0193	0.0193	0.1098	0.1318
C21	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C22	0.107	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0504	0.0391	0.0391	0.0391	0.0391
C23	0.0695	0.0254	0.0254	0.0254	0.0254	0.0254	0.0254	0.1866	0.0254	0.0254	0.1234	0.1234
C24	0.0662	0.0242	0.0242	0.0242	0.0242	0.0242	0.0242	0.2135	0.0242	0.0242	0.1176	0.1295

Table A.12: Usability-based coupling matrix II of package java.awt.event.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17
C1	0.3788	0.0303	0.2121	0.0152	0.0152	0.1818	0.0152	0.0152	0.0152	0.0152	0.0152	0.0152	0.0152	0.0152	0.0152	0.0152	0.0152
C2	0.0947	0.4421	0.053	0.0038	0.3193	0.0455	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038	0.0038
C3	0.1725	0.0138	0.2709	0.0069	0.0069	0.2334	0.0269	0.0269	0.0269	0.0269	0.0269	0.0269	0.0269	0.0269	0.0269	0.0269	0.0269
C4	0.0358	0.0029	0.02	0.9069	0.0014	0.0172	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014
C5	0.0649	0.4378	0.0364	0.0026	0.3985	0.0312	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026
C6	0.159	0.0127	0.2511	0.0064	0.0064	0.2529	0.0283	0.0283	0.0283	0.0283	0.0283	0.0283	0.0283	0.0283	0.0283	0.0283	0.0283
C7	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.1184	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384
C8	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.1184	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384
C9	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.1184	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384
C10	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.0384	0.1184	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384
C11	0.0891	0.0071	0.1944	0.0036	0.0036	0.1905	0.0376	0.0376	0.0376	0.0376	0.1356	0.0376	0.0376	0.0376	0.0376	0.0376	0.0376
C12	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.0384	0.0384	0.0384	0.1184	0.0384	0.0384	0.0384	0.0384	0.0384
C13	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.1184	0.0384	0.0384	0.0384	0.0384
C14	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.1184	0.0384	0.0384	0.0384
C15	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.1184	0.0384	0.0384
C16	0.0928	0.0074	0.2024	0.0037	0.0037	0.1983	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.0391	0.1004	0.0391
C17	0.0909	0.0073	0.1983	0.0036	0.0036	0.1943	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.0384	0.1184

Table A.13: Coupling matrix of package uci.uml.checklist.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17
C1	0.641	0	0.359	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3	0.1136	0	0.417	0	0	0.3114	0.0144	0.0144	0.0144	0.0144	0.0144	0.0144	0.0144	0.0144	0.0144	0.0144	0.0144
C4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
C5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
C6	0	0	0.3565	0	0	0.4052	0.0217	0.0217	0.0217	0.0217	0.0217	0.0217	0.0217	0.0217	0.0217	0.0217	0.0217
C7	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C8	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C9	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C10	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C11	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C12	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C13	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C14	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C15	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C16	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311
C17	0	0	0.2838	0	0	0.3737	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311	0.0311

Table A.14: Usability-based coupling matrix of package uci.uml.checklist.

APPENDIX B

(Tables definition of the central metrics repository)

Table: System

Column	Type	Size
SystemName	Text	50

Primary Key: SystemName

Table: Class

Column	Type	Size
ClassNumber	Number (Long)	4
SystemName	Text	50
ClassName	Text	50
IsAbstract	Yes/No	1

Primary Key: ClassNumber *(Note: this attribute was introduced for efficiency)*
Foreign Key: SystemName ***references*** System (SystemName)

Table: DataType

Column	Type	Size
TypeName	Text	50
TypeWeight	Number (Long)	4

Primary Key: TypeName

Table: Attribute

Column	Type	Size
ClassNumber	Number (Long)	4
AttributeName	Text	50
Type	Text	50
Modifier	Text	50

Primary Key: ClassNumber + AttributeName
Foreign Key: ClassNumber ***references*** Class (ClassNumber)
Foreign Key: Type ***references*** DataType (TypeName)

Table: Method

Column	Type	Size
ClassNumber	Number (Long)	4
MethodName	Text	50
Signature	Text	100
Modifier	Text	50
LOC	Number (Long)	4
IsAbstract	Yes/No	1
ReturnedType	Text	50

Primary Key: ClassNumber + MethodName + Signature
Foreign Key: ClassNumber ***references*** Class (ClassNumber)
Foreign Key: ReturnedType ***references*** DataType (TypeName)

Table: Parameter

Column	Type	Size
ClassNumber	Number (Long)	4
MethodName	Text	50
Signature	Text	100
ParameterName	Text	50
Type	Text	50

Primary Key: ClassNumber + MethodName + Signature + ParameterName
Foreign Key: ClassNumber + MethodName + Signature
references Method (ClassNumber, MethodName, Signature)
Foreign Key: Type ***references*** DataType (TypeName)

Table: ParentOf

Column	Type	Size
ClassNumber	Number (Long)	4
ParentClassNumber	Number (Long)	4

Primary Key: ClassNumber + ParentClassNumber
Foreign Key: ClassNumber ***references*** Class (ClassNumber)
Foreign Key: ParentClassNumber ***references*** Class (ClassNumber)

Table: UsedAttributes

Column	Type	Size
ClassNumber	Number (Long)	4
AncestorClassNumber	Number (Long)	4
AttributeName	Text	50
Frequency	Number (Long)	4

Primary Key: ClassNumber + AncestorClassNumber + AttributeName
Foreign Key: AncestorClassNumber + AttributeName
references Attribute (ClassNumber, AttributeName)
Foreign Key: ClassNumber ***references*** Class (ClassNumber)

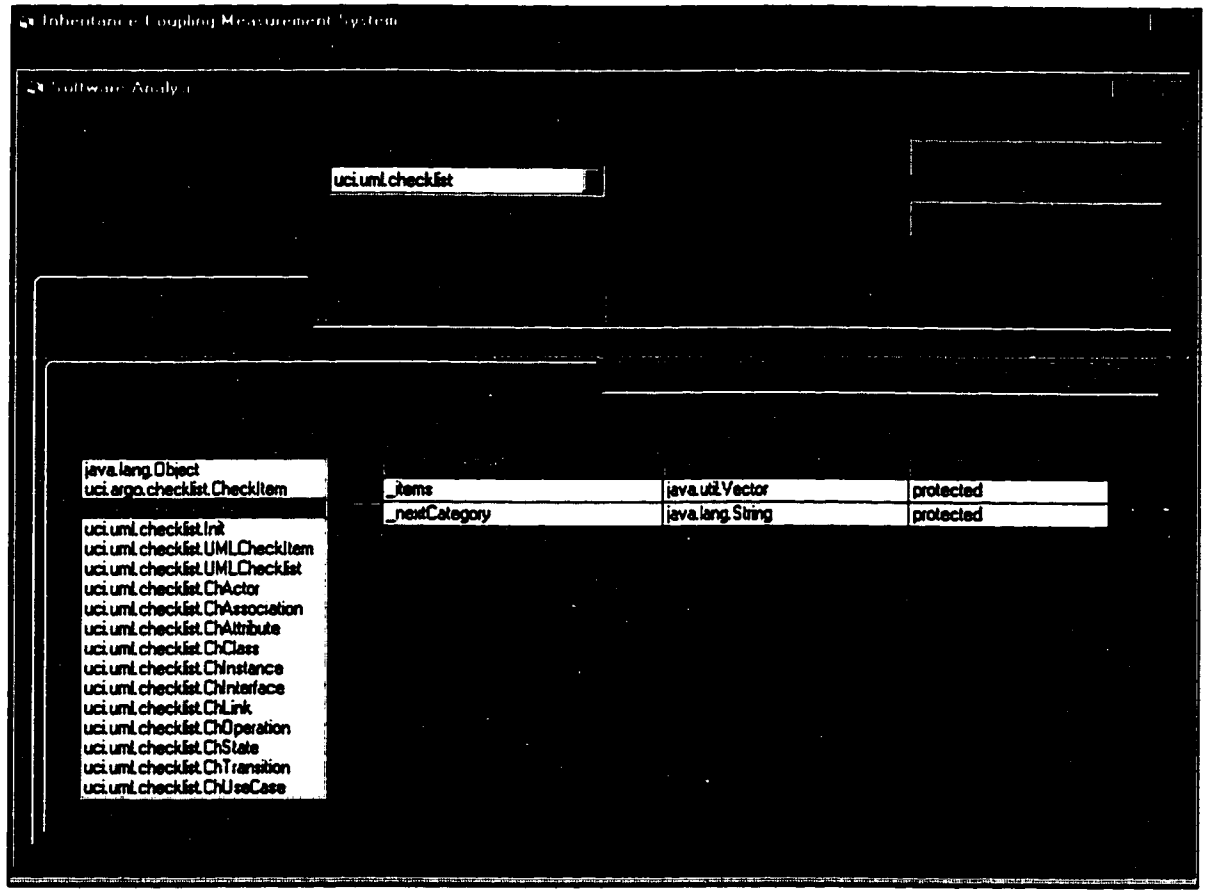
Table: UsedMethods

Column	Type	Size
ClassNumber	Number (Long)	4
AncestorClassNumber	Number (Long)	4
MethodName	Text	50
Signature	Text	100
Frequency	Number (Long)	4

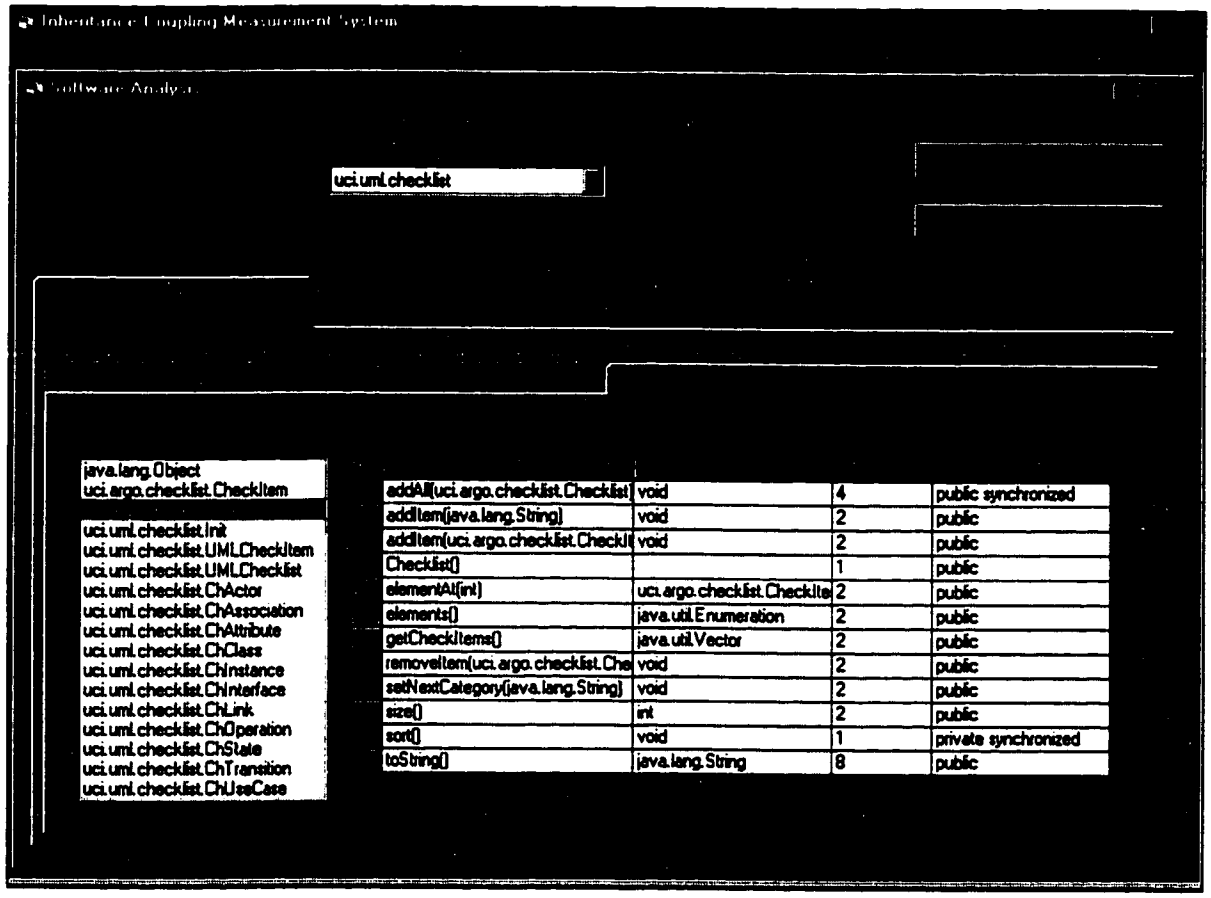
Primary Key: ClassNumber + AncestorClassNumber + MethodName + Signature
Foreign Key: AncestorClassNumber + MethodName + Signature
references Method (ClassNumber, MethodName, Signature)
Foreign Key: ClassNumber ***references*** Class (ClassNumber)

APPENDIX C

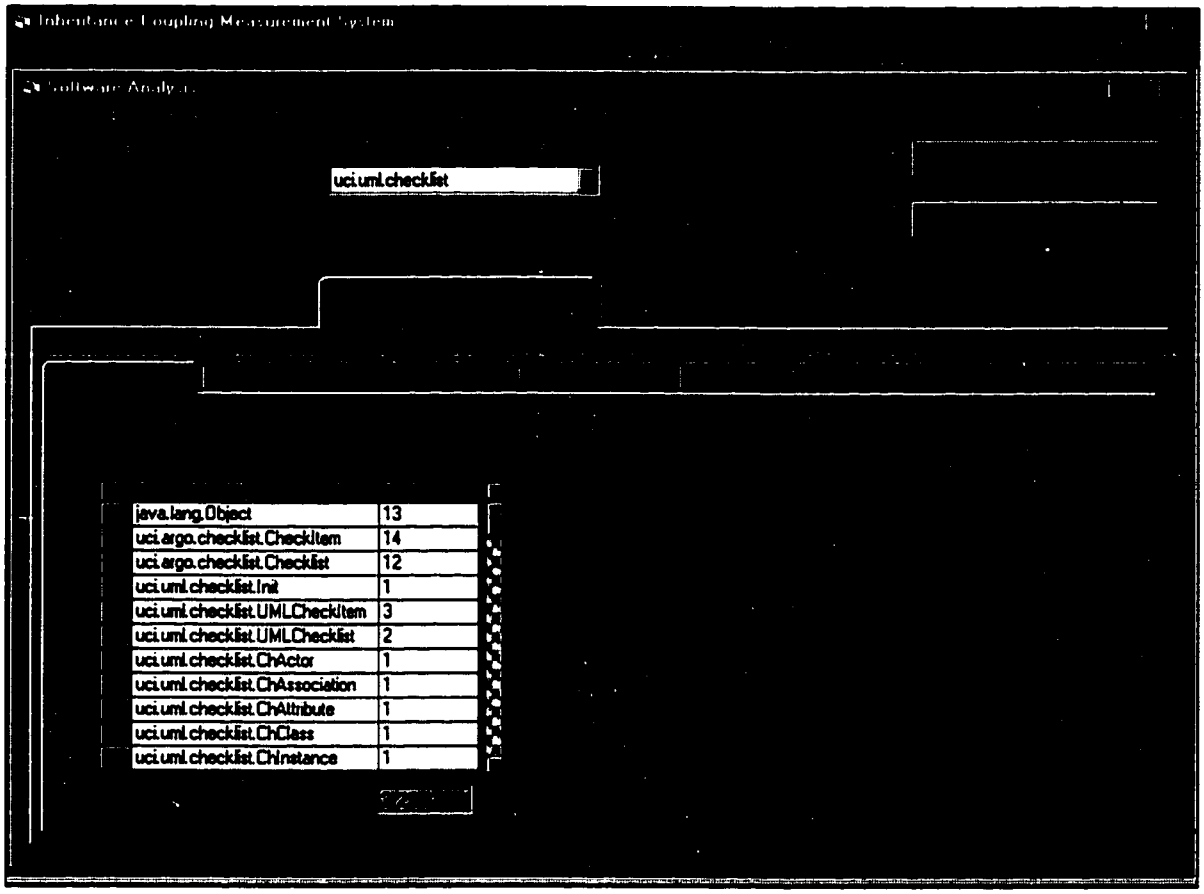
(User Interfaces of the developed tool)



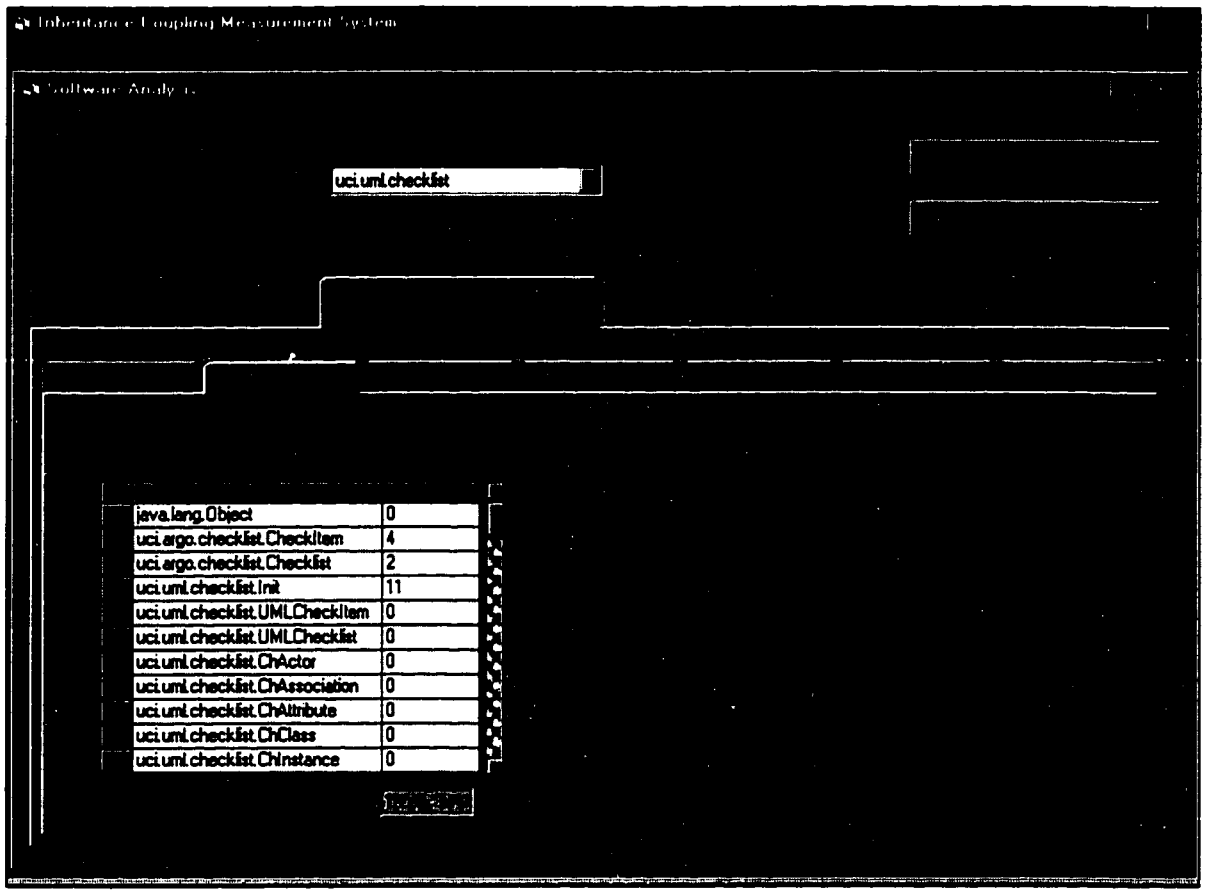
This screen views detailed information about all attributes defined in each class of the selected system. For each attribute, it provides its name, type and modifier. By clicking on a class name from the classes list, information about all attributes defined in that class will be shown.



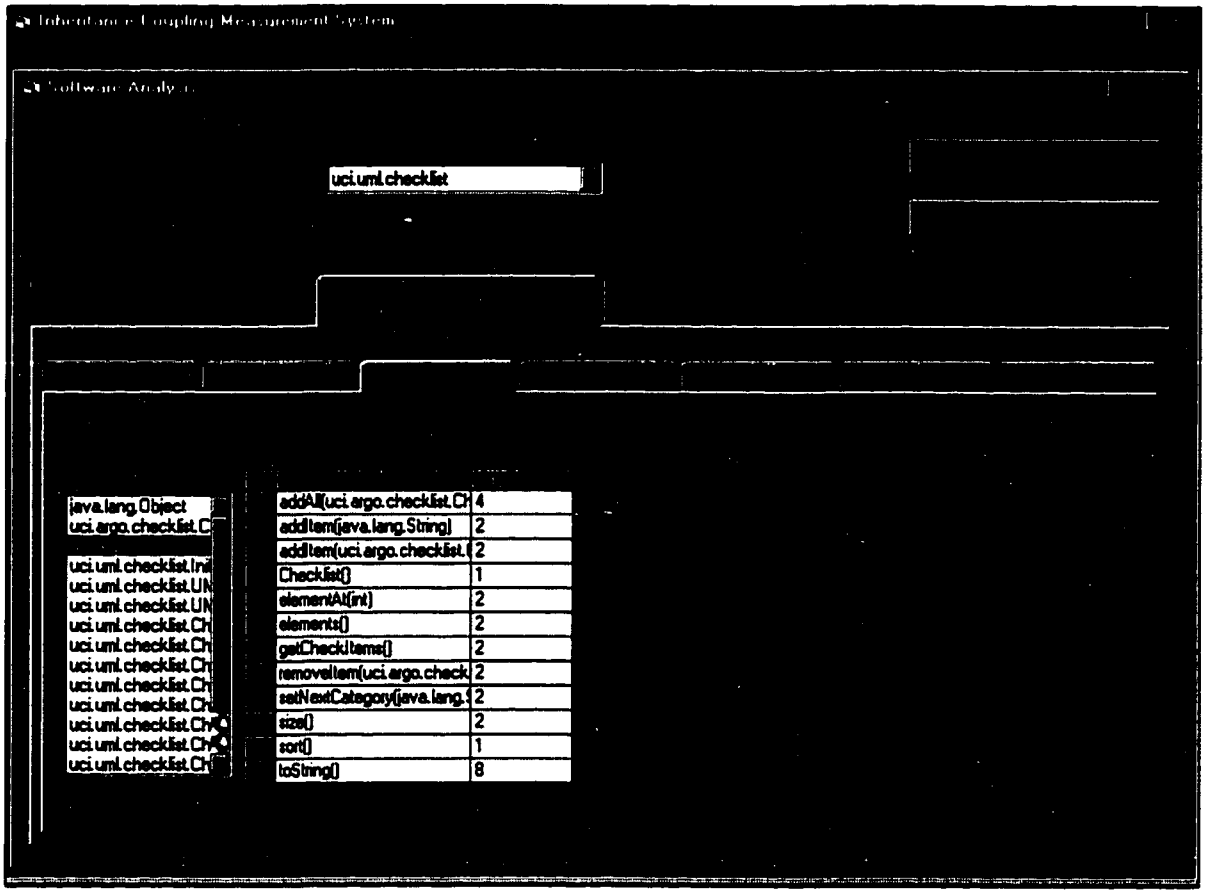
This screen views detailed information about all methods defined in each class of the selected system. For each method, it provides its signature, returned type, line-of-code and modifier. By clicking on a class name from the classes list, information about all methods defined in that class will be shown.



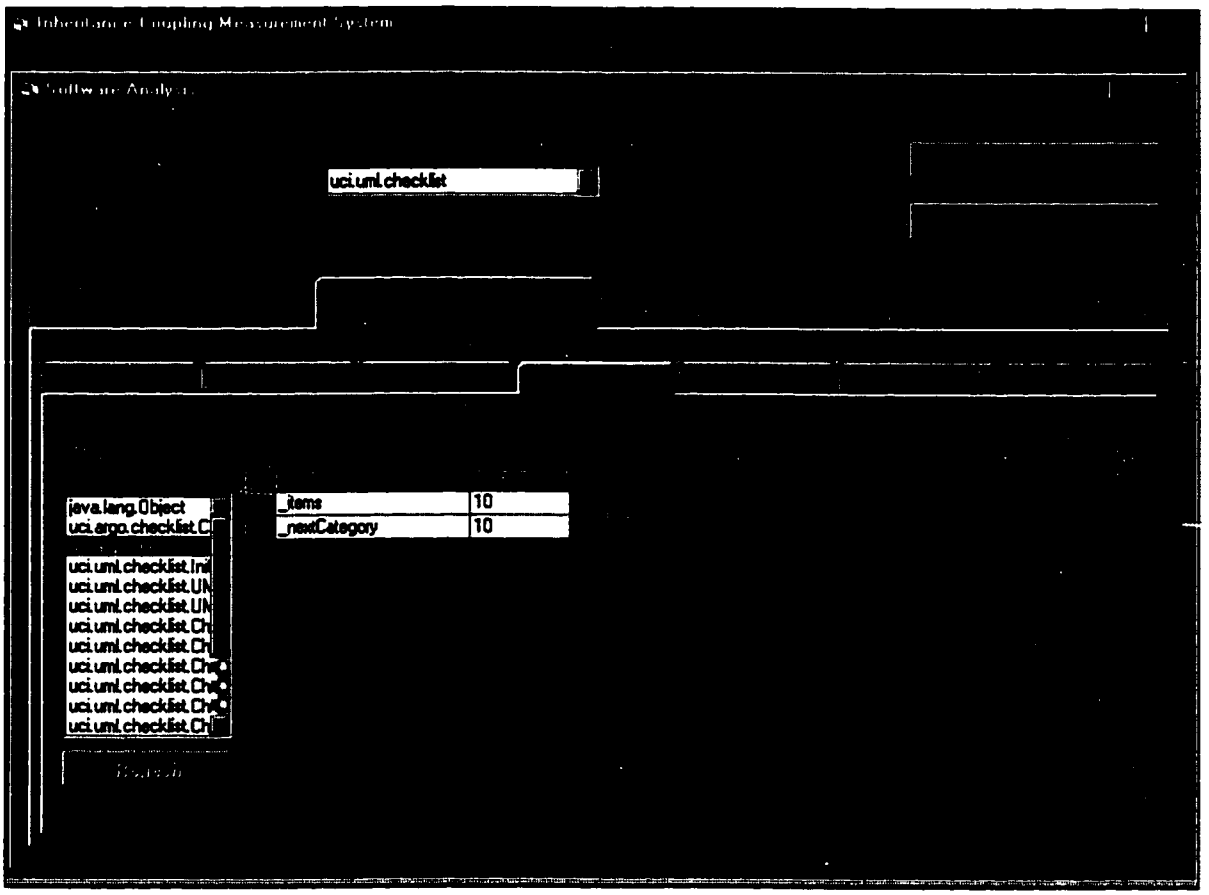
This screen displays the result of applying number of methods per class (NOM) metric against the selected system. It also provides chart view of the metric result.



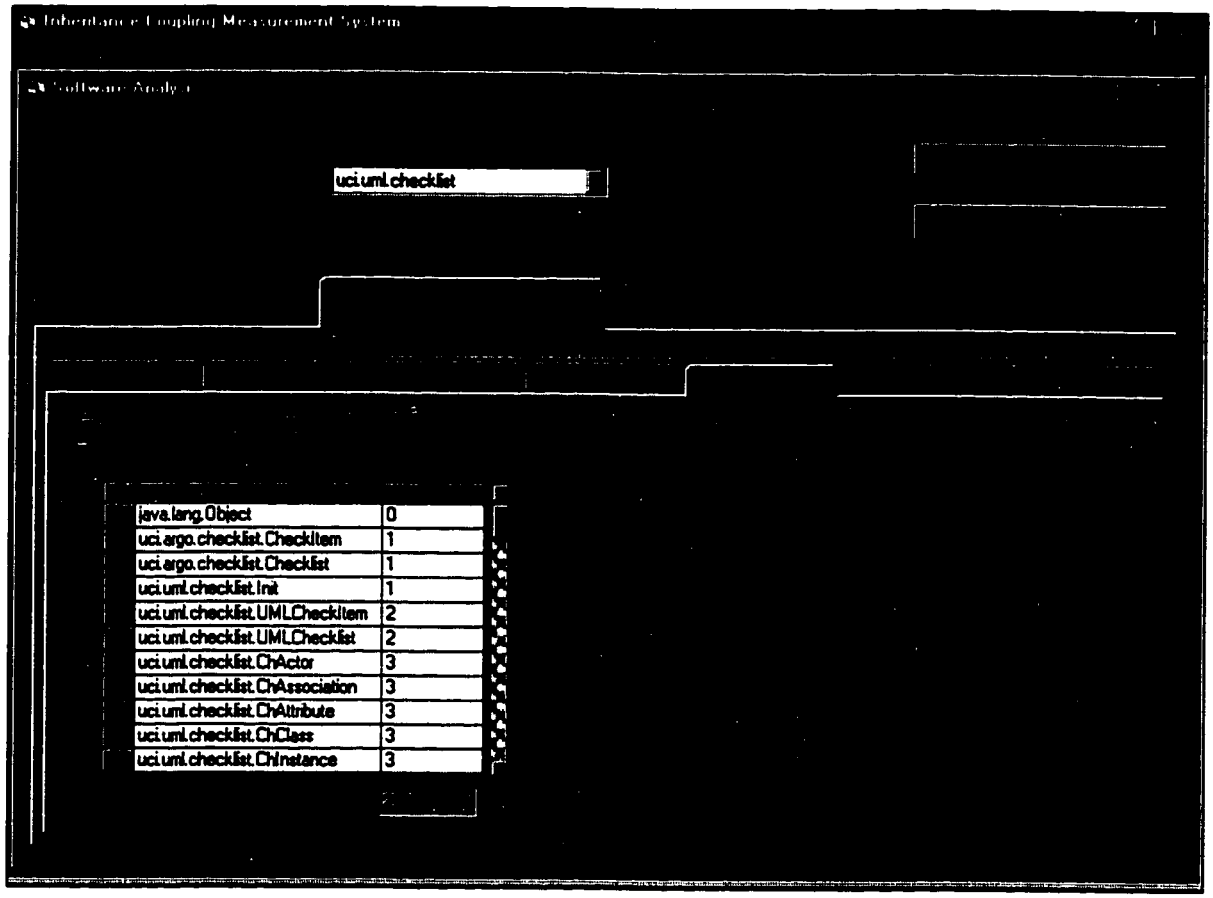
This screen displays the result of applying number of attributes per class (NOA) metric against the selected system. It also provides chart view of the metric result.



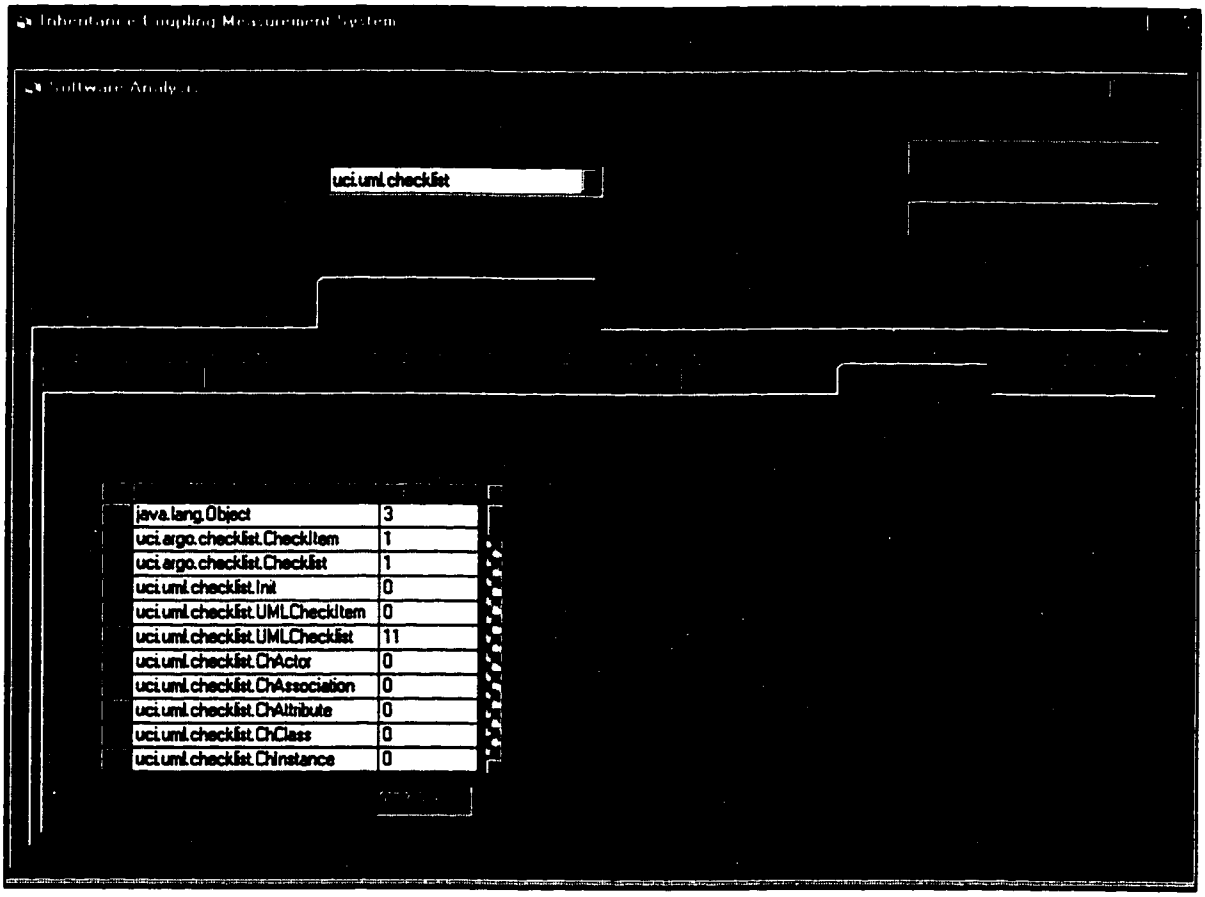
This screen views the result of applying weight of methods per class (WOM) metric against the selected system. It also provides chart view of the metric result. By clicking on a class name from the classes list, line-of-code (LOC) for each method defined in that class will be shown.



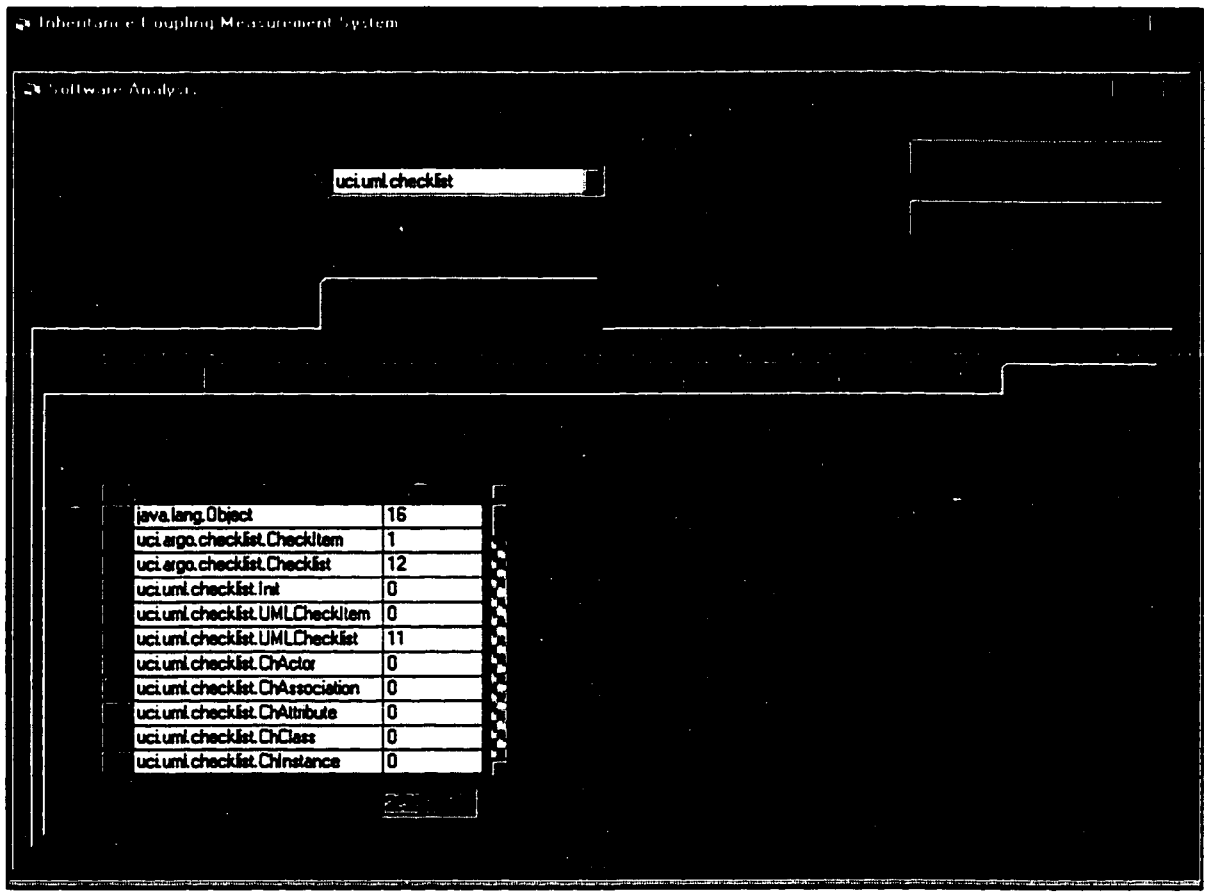
This screen views the result of applying weight of attributes per class (WOA) metric against the selected system. It also provides chart view of the metric result. By clicking on a class name from the classes list, the weight of each attribute defined in that class will be shown.



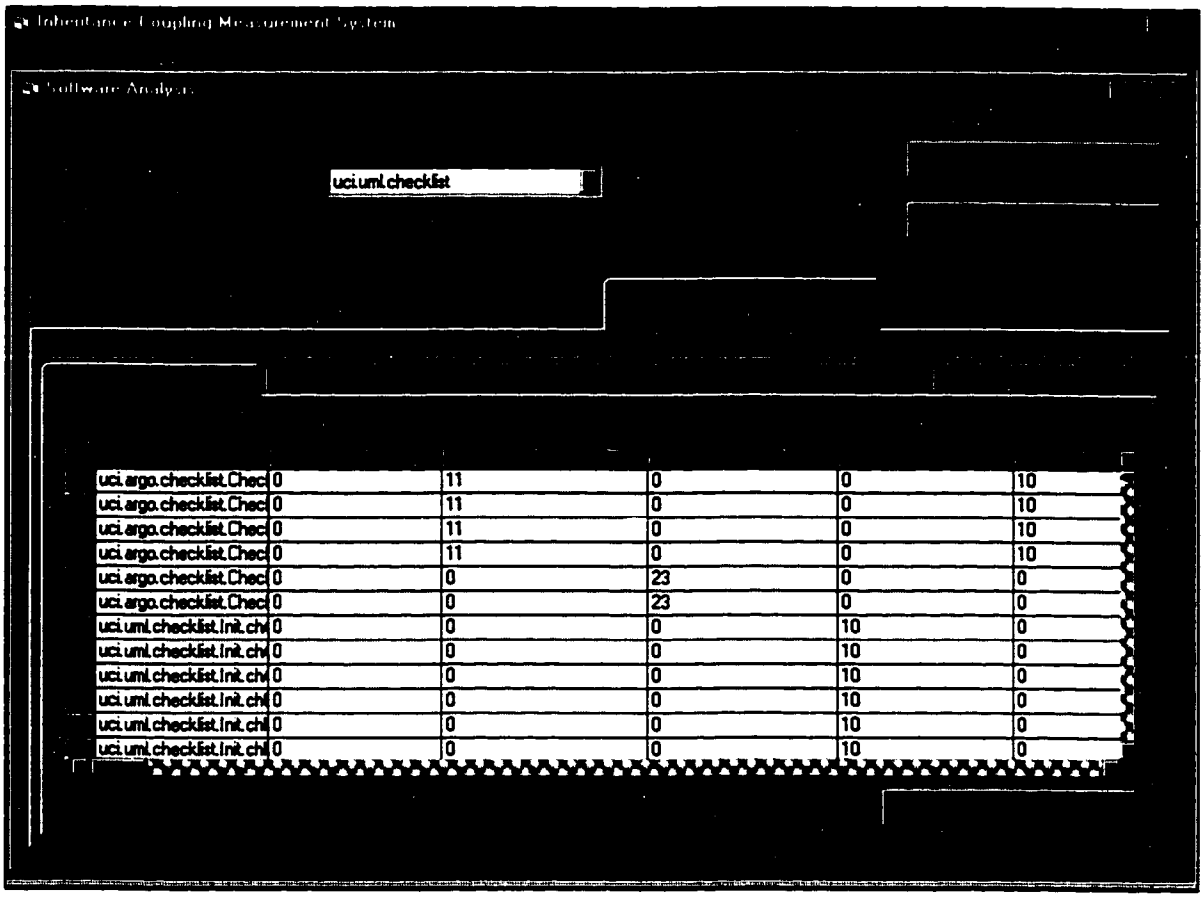
This screen displays the result of applying depth of inheritance tree (DIT) metric against the selected system. It also provides chart view of the metric result.



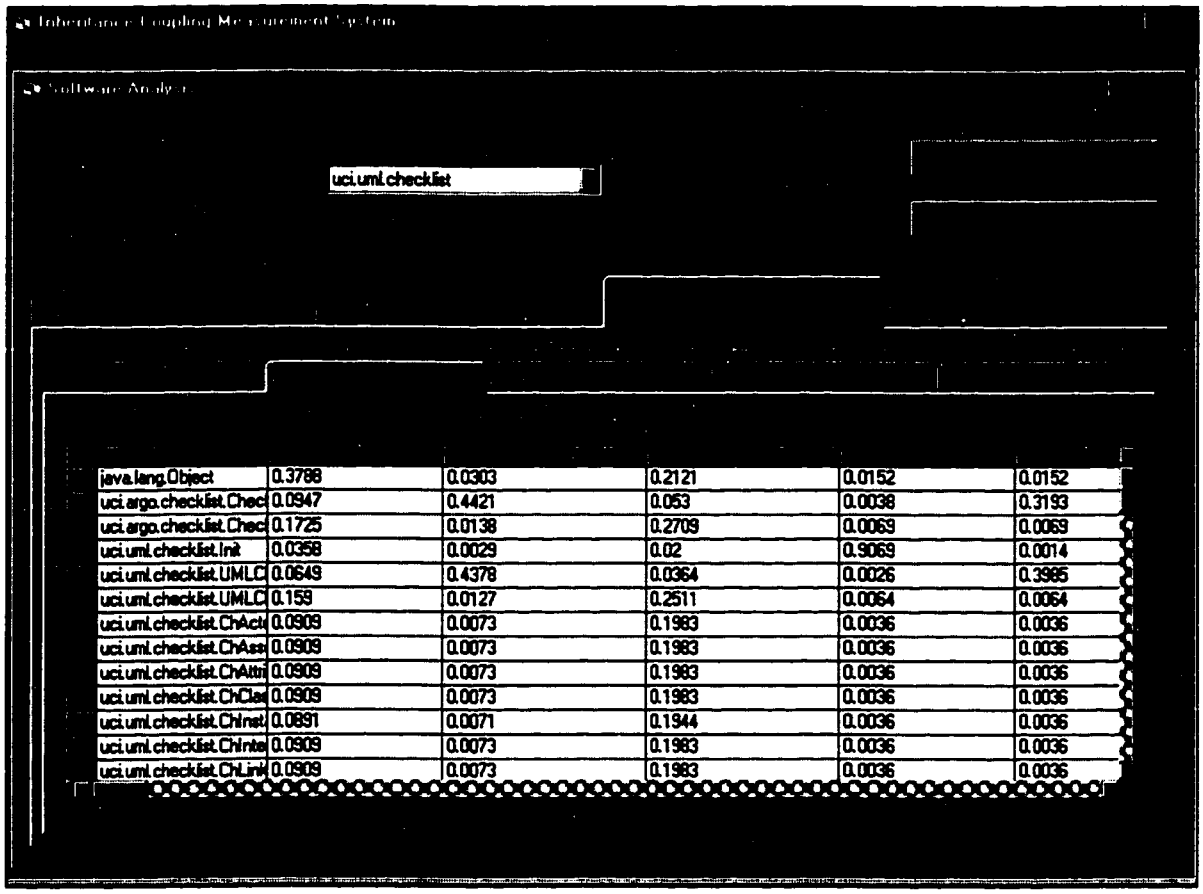
This screen displays the result of applying number of children (NOC) metric against the selected system. It also provides chart view of the metric result.



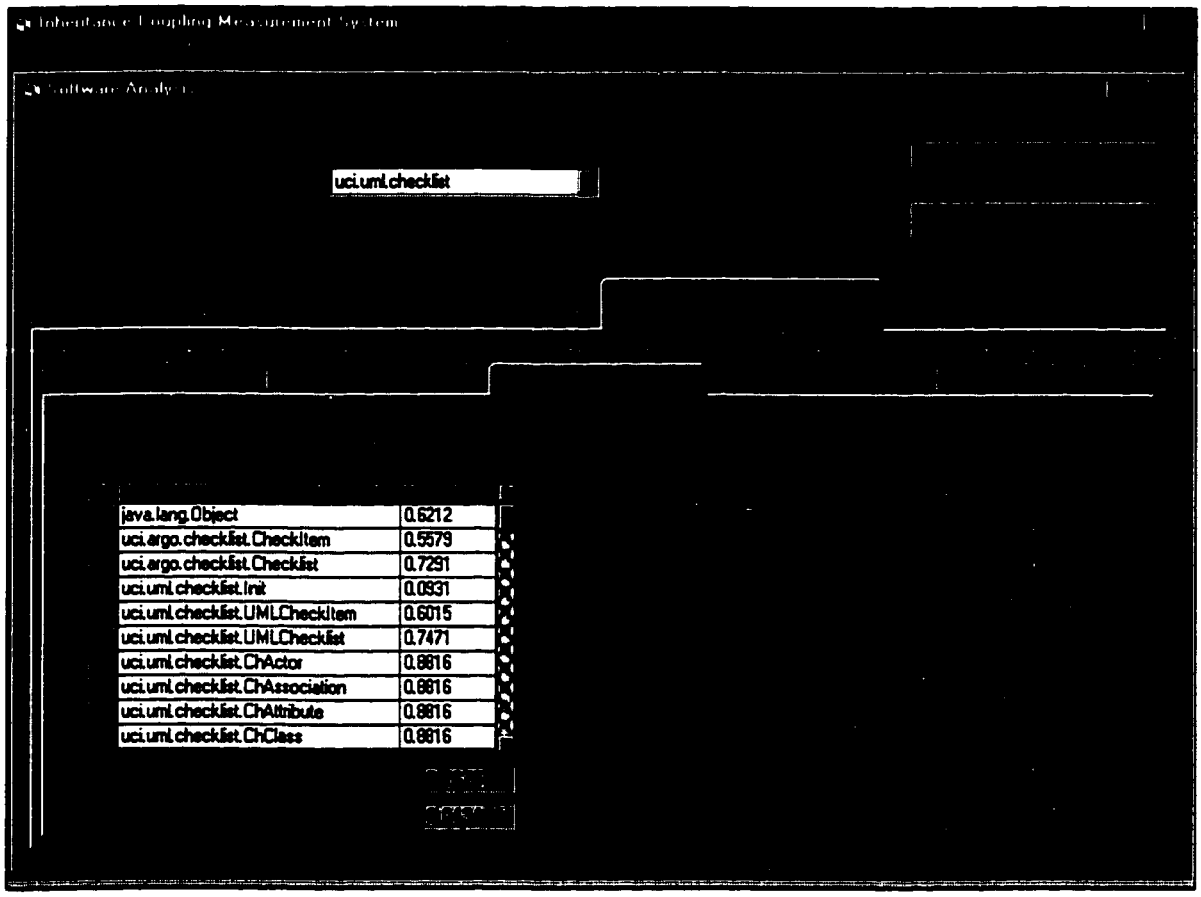
This screen displays the result of applying number of descendants (NOD) metric against the selected system. It also provides chart view of the metric result.



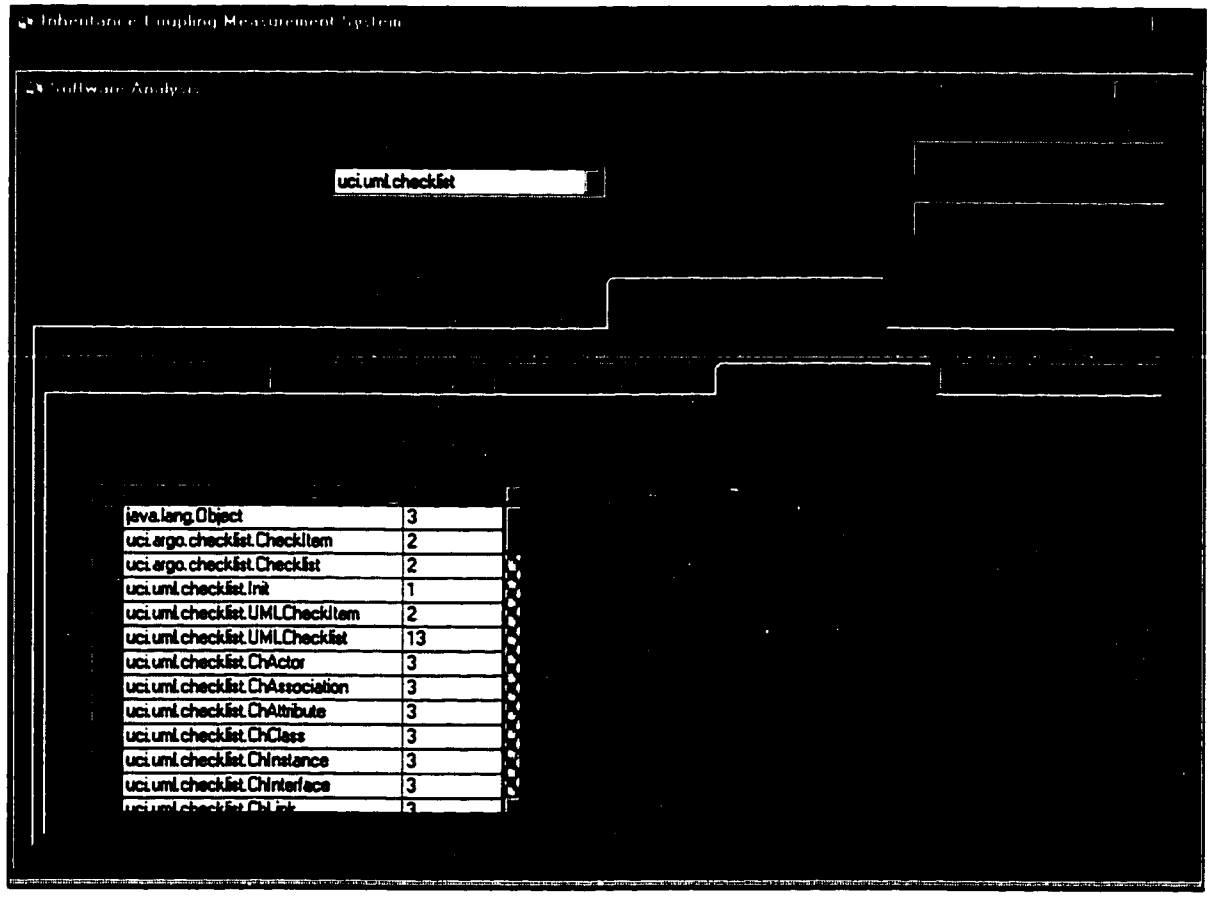
This screen views the definition matrix of the inheritance coupling metric proposed by AlMulla.



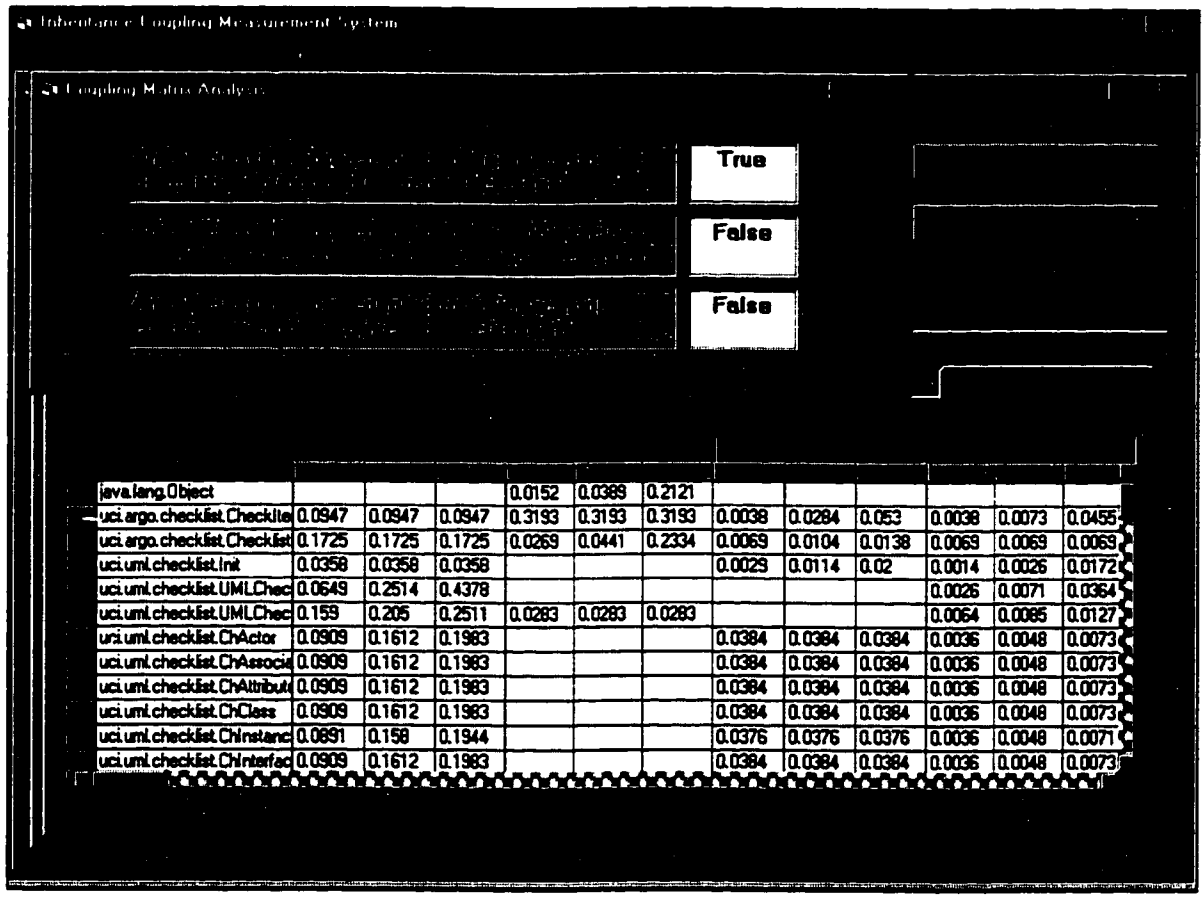
This screen views the coupling matrix of the inheritance coupling metric proposed by AlMulla. Each entry C_{ij} of the coupling matrix represents the extent to which class i is coupled to class j. This is known as class-to-class coupling (CTC).



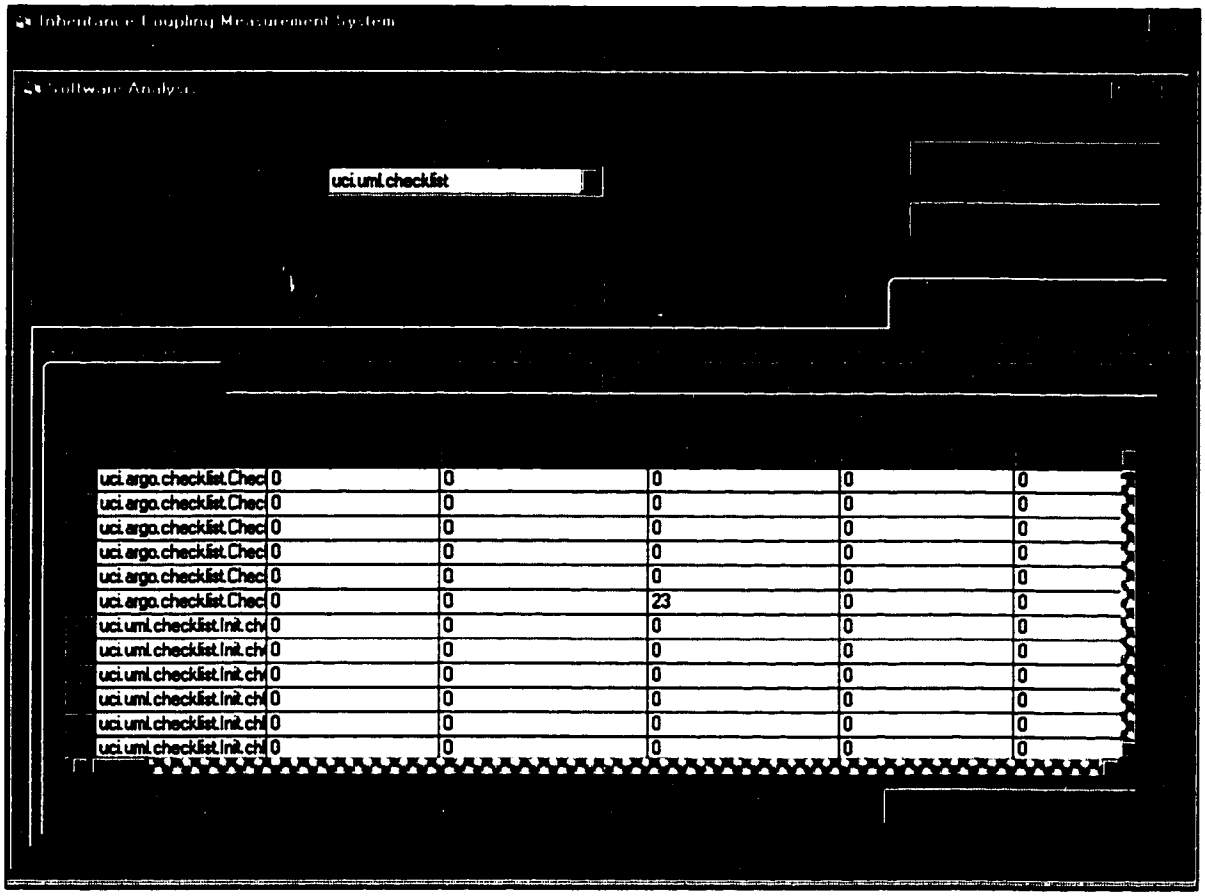
This screen displays the result of applying individual class coupling (ICC) metric and overall system coupling (OSC) metric proposed by AlMulla against the selected system. It also provides chart view of the metrics result.



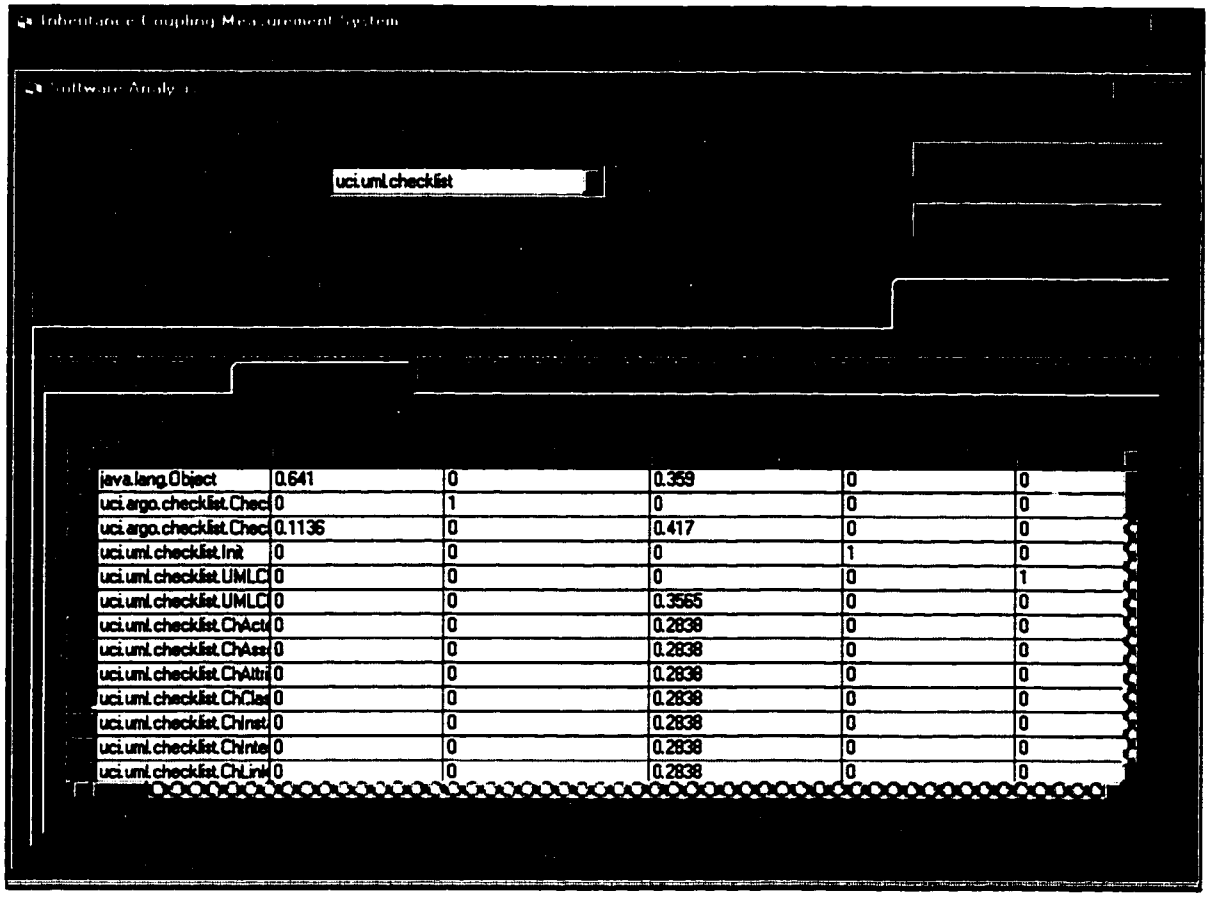
This screen displays the result of applying coupling through inheritance metric proposed by Henry and Li against the selected system. It also provides chart view of the metric result.



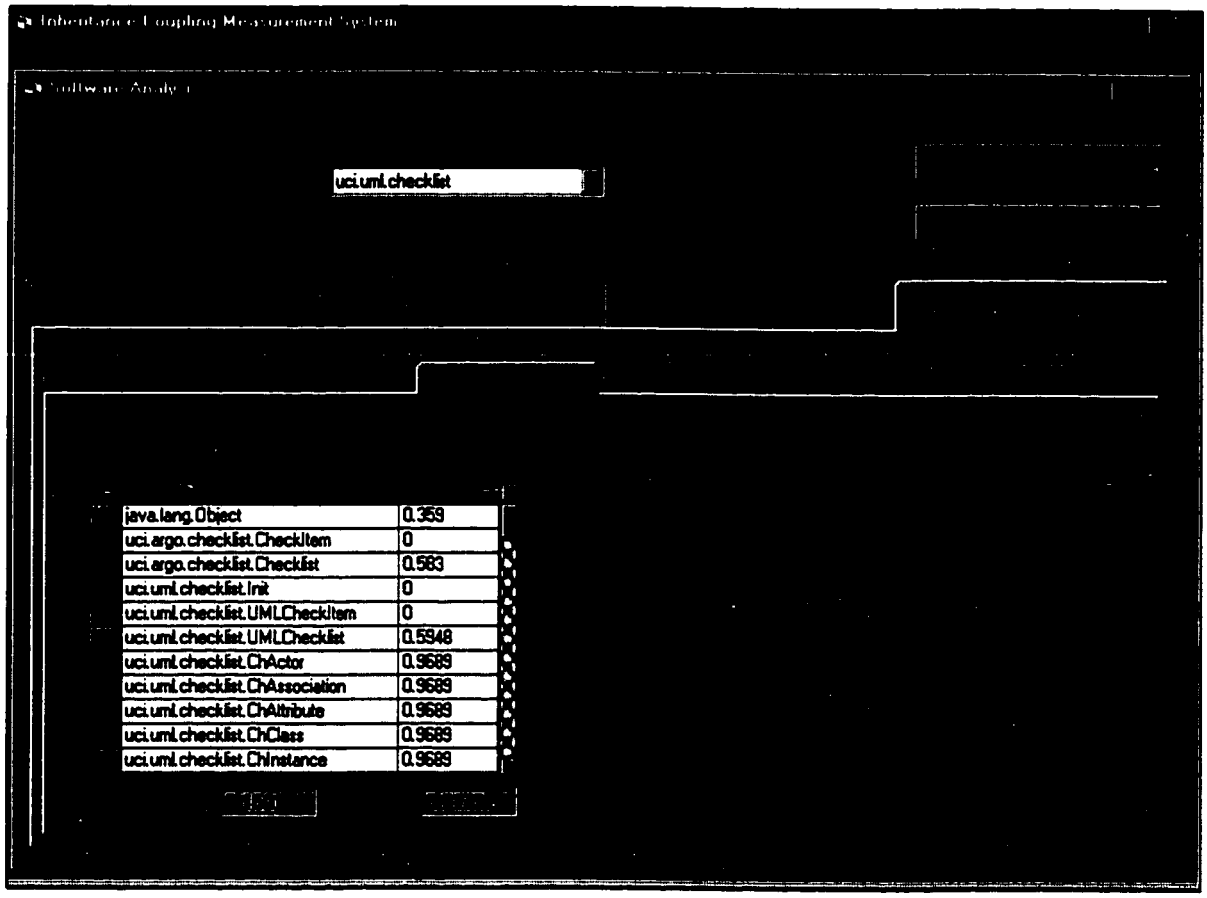
This screen views statistical information about the coupling matrix of AlMulla metric. It gives the minimum, average, and maximum coupling values between each class in the selected system and its ancestor, descendant, sibling, and other classes. In addition, the coupling matrix is tested against the above three questions.



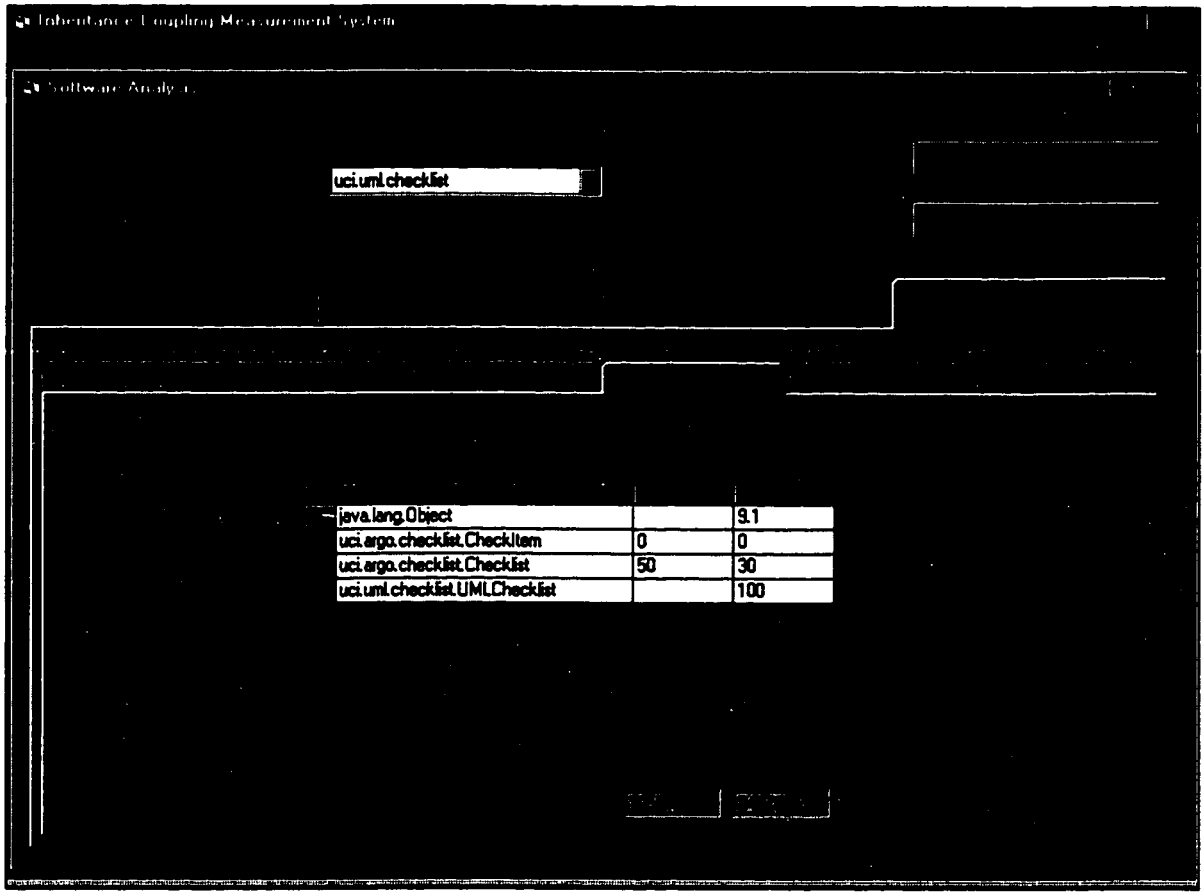
This screen views the definition matrix of the usability-based inheritance coupling metric.



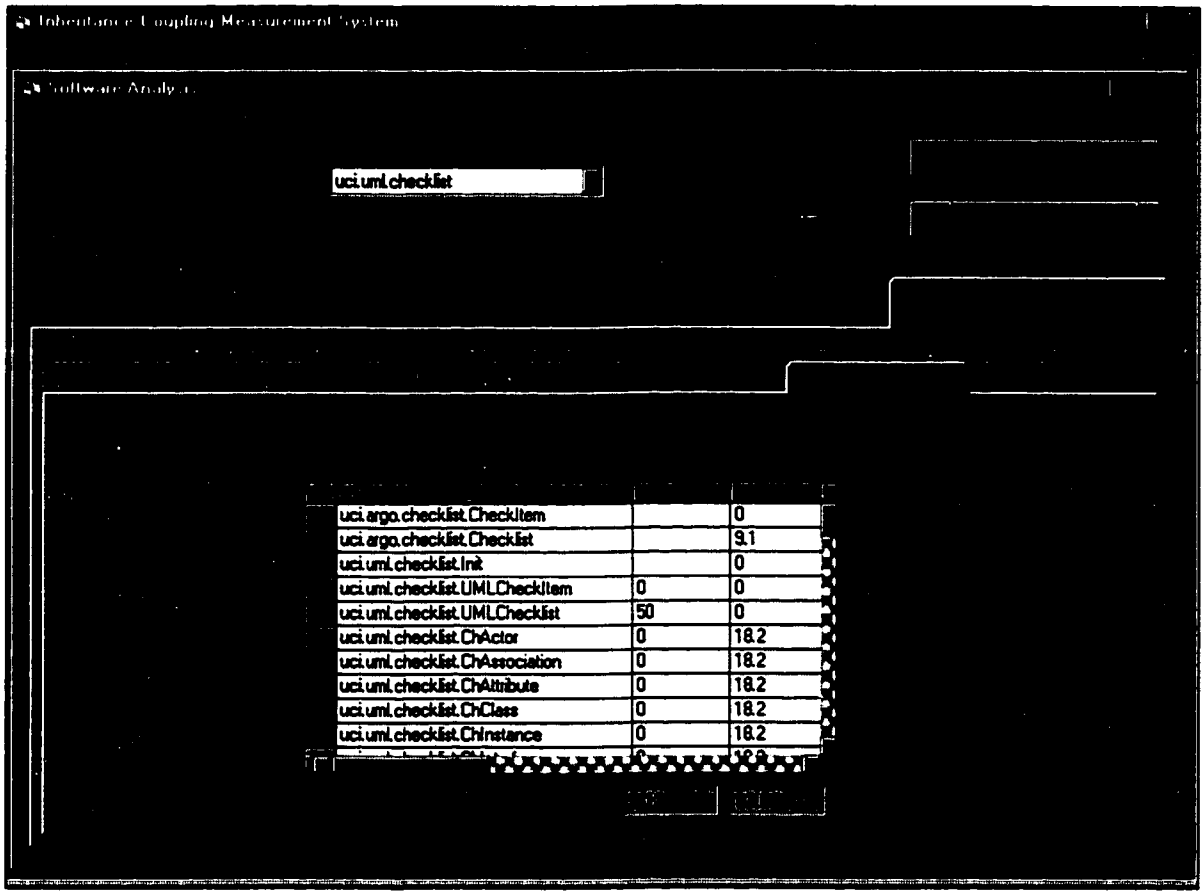
This screen views the coupling matrix of the usability-based inheritance coupling metric. Each entry UC_{ij} of the coupling matrix represents the extent to which class i is coupled to class j. This is known as usability-based class-to-class coupling (UCTC).



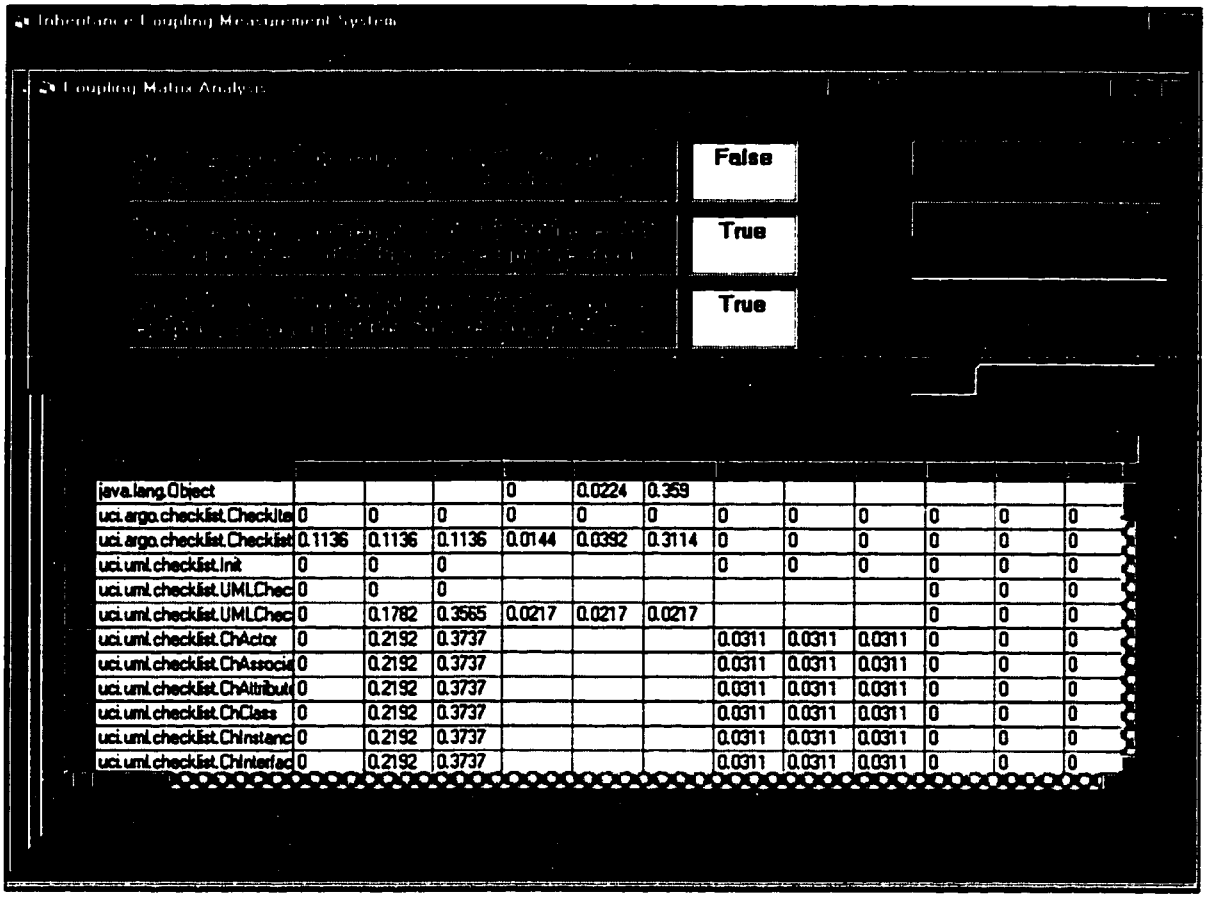
This screen displays the result of applying usability-based individual class coupling (UICC) and usability-based overall system coupling (UOSC) metrics against the selected system. It also provides chart view of the metrics result.



This screen views the percentage of inherited attributes and methods from each non-leaf class in the system that are used by any descendant class(s) to those that are not used.



This screen views the percentage of inherited attributes and methods by each non-root class in the system that are used by it to those that are not used.



This screen views statistical information about the coupling matrix of the usability-based inheritance coupling metric. It gives the minimum, average, and maximum coupling values between each class in the selected system and its ancestor, descendant, sibling, and other classes. In addition, the coupling matrix is tested against the above three questions.

REFERENCES

[Algh94] Jarallah AlGhamdi, *Programming Languages: A Quantitative Methodology for Assessments Software Metrics to Measure Syntactic Properties*, Ph.D. Thesis, Arizona State University, Aug. 1994.

[Almu98] Ahmed Al-Mulla, *Measuring Class Coupling in Object-Oriented Design*, Master Thesis, KFUPM, Mar. 1998.

[Argo] Argo/UML: Free Object-Oriented Design Tool available at http://www.ics.uci.edu/pub/arch/uml/index.html.

[Bria99] Lionel Briand, John Daly and Jurgen Wust, *"A Unified Framework for Coupling Measurement in Object-Oriented Systems"*, IEEE Transactions on Software Engineering, Vol. 25, No. 1, pp. 91-121, Jan/Feb 1999.

[Bris96] Sharmila Bristol, *"Tools for Object-Oriented (OO) Metrics Collection"*, OOPSLA '96-Workshop: OO Product Metrics, 1996,

[Broo94] Christopher Brooks and Christopher Buell, *"A Tool for Automatically Gathering Object-Oriented Metrics"*, Proceedings of the IEEE 1994 National Aerospace and Electronics Conference (NAECON 1994), vol. 2, pp. 835-838, 1994.

[Bucc98] G. Bucci, F. Fioravanti, P. Nesi and S. Perlini, *"Metrics and Tool for System Assessment"*, Proceedings of 4th IEEE International Conference on Engineering of Complex Systems (ICECCS 98), pp. 36-46, 1998.

[Budd91] Timothy Budd, *An Introduction to Object-Oriented Programming*, Addison-Wesley, 1991.

[Can84] Fazli Can and Esen Ozkarahan, *"Two Partitioning Type Clustering Algorithms"*, Journal of the American Society for Information Science, Vol. 35, No. 5, pp. 268-276, 1984.

[Can90] Fazli Can and Esen Ozkarahan, *"Concepts and Effectiveness of the Cover-Coefficient-Based Clustering Methodology for Text Databases"*, ACM Transactions on Database Systems, vol. 15, no. 4, pp. 483-517, Dec. 1990.

[Chen93] J-Y Chen and J-F Lu, *"A New Metric for Object-Oriented Design"*, Information and Software Technology, pp. 232-240, June 1993.

[Chid91] Shyam Chidamber and Chris Kemerer, *"Towards a Metrics Suite for Object Oriented Design"*, A. Paepcke, ed., Proc. Conf. Object-Oriented Programming: Systems, Languages and Applications, OOPSLA 91, Oct. 1991.

[Chid94] Shyam Chidamber and Chris Kemerer, *"A Metrics Suite for Object Oriented Design"*, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp.476-493, June 1994.

[Coad91] Peter Coad and Edward Yourdon, *Object-Oriented Design*, Prentice-Hall, 1991.

[Daly96] John Daly, Andrew Brooks, James Miller, Marc Roper and Murray Wood, *"An Empirical Study Evaluating Depth of Inheritance on the Maintainability of Object-Oriented Software"*, Tech. Rep., University of Strathclyde, Britain, 1996.

[DeMa82] Tom DeMarco, *Controlling Software Projects*, Prentice Hall, New York, 1982.

[Eder94] Johann Eder, Gerti Kappel and Michel Schrefl, *"Coupling and Cohesion in Object-Oriented Systems"*, Technical Report, Univ. of Klagenfurt, Austria, 1994.

[Fent93] Norman Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, First Edition, 1993.

[Fior98] Fabrizio Fioravanti, Paolo Nesi and Sandro Perlini, *"A Tool for Process and Product Assessment of C++ Applications"*, Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering, pp. 89-95, 1998.

[Hend96] Brain Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, Prentice Hall PTR, 1996.

[Heri98] Marjan Hericko, Ivan Rozman, Romana Horvat, Tomaz Domajnko and Jozsef Gyorkos, *"OO Metrics Data Gathering Environment"*, Proceedings of Technology of Object-Oriented Languages (TOOLS 24), pp. 80-85, 1998.

[JDK1.2] *Java Development Kit 1.2 Documentation*, Sun Microsystems, 1998, available at http://java.sun.com/products/jdk/1.2/index.html.

[Kim89] Won Kim and Frederick Lochovsky, *Object-Oriented Concepts, Databases, and Applications*, ACM Press, 1989.

[Kors90] Tim Korson and John McGregor, *"Understanding Object-Oriented: A Unifying Paradigm"*, Communication of the ACM, vol. 33, no. 9, pp. 40-60, Sep. 1990

[Kuo94] Y. Kuo, *"When to Inherit and When Not to"*, Proceedings of Software Engineering Conference, pp. 378-387, 1994.

[Li93] W. Li and S. Henry, *"Object-Oriented Metrics that Predict Maintainability"*, Journal of Systems and Software, vol. 23, no. 2, pp. 111-122, 1993.

[Lore93] M. Lorenz, *Object-Oriented Software Development: A Practical Guide*, Prentice Hall, NJ, 1993.

[Lore94] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice Hall, NJ, 1994.

[Mose97] Simon Moser and Vojislav Misic, *"Measuring Class Coupling and Cohesion: A Formal Metamodel Approach"*, Proceedings of International Computer Science Conference (ICSC 97), pp.31-40, 1997.

[Myer78] G. Myers, *Composite/Structured Design*, Van Nostrand Reinhold, 1978.

[Rahm99] Dan Rahmel, *Sams Teach Yourself Database Programming with Visual Basic 6 in 24 Hours*, Sams, 1999.

[Rose98] Linda Rosenberg, *"Applying and Interpreting Object-Oriented Metrics"*, Software Technology Conference, Utah, Apr. 1998.

[Sebe93] Robert Sebesta, *Concepts of Programming Languages*, Benjamin/Cummings, Second Edition, 1993.

[Snee95] Harry Sneed, *"Understanding Software Through Numbers: a Metric Based Approach to Program Comprehension"*, Journal of Software Maintenance: Research and Practice, vol. 7, pp. 405-419, 1995.

[Somm96] Ian Sommerville, *Software Engineering*, Addison-Wesley, Fifth Edition, 1996.

[Stev74] W. Stevens, G. Myers and L. Constantine, *"Structured Design"*, IBM Systems Journal, vol. 13, no. 2, pp. 115-139, 1974.